Dissertation zur Erlangung des Doktorgrads der Fakultät für Angewandte Wissenschaften der Albert-Ludwigs-Universität Freiburg im Breisgau

Free Surface Flow and the IMTEK Mathematica Supplement

Oliver Rübenkönig

VERSION: 1.1
Freiburg – Leipzig – Frankfurt a. M.

Autor:	Oliver Rübenkönig
Titel:	Free Surface Flow and the IMTEK Mathematica Supplement
Adresse:	Lehrstuhl für Simulation Institut für Mikrosystemtechnik (IMTEK) Albert-Ludwigs-Universität Freiburg Georges-Köhler-Allee 103 79110 Freiburg Germany
Dekan:	Prof. Dr. Hans Zappe
Referenten:	Prof. Dr. Jan G. Korvink Prof. Dr. Peter Woias
Datum:	18.12.2008

Zusammengestohlen aus Verschiedenem diesem und jenem.

(Ludwig van Beethoven)

Contents

1 Introduction **5**

1.1 Objectives of the Thesis 6

1.2 Overview 6

1.3 Major Results 7

1.4 Document license 9

2 Modelling with Operators **11**

2.1 Nodes, Elements & their Graphs 12

2.2 Systems of Equations 13

2.3 From Graphs to Systems 14

2.4 Illustrations 15

3 Modelling with Lumped Operators **21**

3.1 Lagrangian Dynamics 22

3.2 Modified Nodal Approach 25

3.3 Lagrangian Modified Nodal Approach 25

4 Modelling with Partial Differential Equation Operators **29**

4.1 Building Blocks 30

4.2 Combining Operators to Model Partial Differential Equations 34

5 Modelling Free Surface Flow with Operators **43**

5.1 Flow: The Navier-Stokes Equation 44

5.2 Free Surfaces: The Level-Set Method 47

5.3 Coupling Components 51

6 Operator Discretisation **53**

6.1 Lumped Operators 54

6.2 Weighted Residual Method 58

6.3 Finite Element Method 62

6.4 Initial & Boundary Conditions 85

7 Implementation **91**
7.1 Design Philosophy 92
7.2 Program Flow 94
7.3 Graphs 95
7.4 Operators & Matrix Assembly 101
7.5 Boundary Conditions 109
7.6 Systems 112
7.7 System Analysis 113

8 Application Examples **119**
8.1 Verification of the Code 120
8.2 Test Cases 122
8.3 The Ink-Jet Model 135
8.4 Final Remarks 138

9 Appendix **143**
9.1 Quadratic Serendipity Shape Functions 143
9.2 Transient Mechanics 143
9.3 Acknowledgements 144

Abstract

The purpose of this study was to develop an open, flexible and extendable framework for ink-jet simulation in *Mathematica*.

The developed software tool allows to inspect and engage in all processes during simulations. With the facility to model different lumped networks and partial differential equations simultaneously and in one software environment we have great flexibility at hand.

The proposed computational approach relies on one basic component: modelling with operators. We identify a methodology to combine, both, lumped networks and mesh based partial differential equation methods into graphs. Subsequently these graphs will be converted to systems of (differential algebraic) equations, which then can be solved with backward difference formulae.

We introduce Lagrangian dynamics and the Modified Nodal Approach for lumped electrical networks. Both methods are combined to derive a Lagrangian Modified Nodal Approach. Based on this combined approach we exemplary model electrical circuits.

For a general and mostly automatic treatment of partial differential equations we present six fundamental operators and appropriate methodologies with which we are able to solve arbitrary dimensional multi-physics coupled non-linear partial differential equations consisting of up to second order derivatives in space and time.

The presented automatic linearisation process is applied to the Navier-Stokes equation and results in a system of equations including the tangent stiffness matrix and is suitable to be solved with an affine invariant Newton method. The free surface, separating the two immiscible fluids, is presented by an implicit function and advected with a linear Level-Set method. Fitting into the framework we model the surface tension as a volume force.

The discretisation of the operators is derived and the partial differential equations operators are based on the finite element method. The extendable implementation was done based on the functional programming paradigm. Due to the symbolic nature of *Mathematica* the framework allows for symbolic, complex imaginary and/or numeric designs.

Diverse numerical examples with increasing complexity are presented to verify the presented algorithms. The examples including the advection of a circular bubble, the static behaviour of a bubble suspended in a liquid, a range of rising bubbles showing different degrees of skirted bubbles, the rising of an oil bubble bursting through a surface and an ink-jet simulation model are compared to analytical, numerical and experimental data and found to be in good agreement.

With this simple framework complex phenomena such as ink-jet simulations can be described.

Zusammenfassung

Das Ziel dieser Arbeit ist es ein offenes, flexibles und erweiterbares Gerüst für die Ink-Jet Simulation in *Mathematica* zu erstellen.

Das entstandene Softwarewerkzeug ermöglicht es, zu jedem Zeitpunkt der Simulation alle Prozesse zu überprüfen und mit denselben in Wechselwirkung zu treten. Die Möglichkeit, verschiedenartige konzentrierte Netzwerke und partielle Differentialgleichungen gleichzeitig und in derselben Softwareumgebung zu modellieren, bietet große Freiheiten.

Die vorgestellte Methodik beruht auf einem einzigen grundlegenden Konzept: Modellierung mit Operatoren. Wir stellen eine Methode vor, die konzentrierte Netzwerke und partielle Differentialgleichungen in Graphen verbindet. Jene Graphen werden nachfolgend in Systeme von, möglicherweise algebraischen, Differentialgleichungen überführt.

Wir führen Lagrangesche Dynamik und ein Modified Nodal Approach für konzentrierte Netzwerke ein. Beide Methoden werden zu einem Lagrange Modified Nodal Approach zusammengeführt. Aufbauend auf jenen verbundenen Ansatz zeigen wir exemplarisch die Modellierung elektrisalgebraischenchener Netzwerke.

Die allgemeingültige und größtenteils automatische Behandlung von partiellen Differentialgleichungen beruht auf sechs fundamentalen Operatoren. Es werden Methoden vorgestellt, die es ermöglichen, beliebig dimensionale gekoppelte nichtlineare Gleichungen mit bis zu zweiter Ordnung in Raum und Zeit zu lösen.

Der vorgestellte automatische Linearisierungsprozess wird auf die Navier-Stokes Gleichung angewandt und liefert ein System von Gleichungen, welches die Tangentensteifigkeitsmatrix beinhaltet und für dessen Lösung eine affin invariantes Newton Verfahren geeignet ist. Die freie Oberfläche, die zwei nicht mischbare Flüssigkeiten trennt, wird durch eine implizite Funktion dargestellt und mittels der Level-Set Methode transportiert. Die Oberflächenspannung wird als Volumenkraft modelliert.

Die Diskretisierung der Partialdifferentialgleichungsoperatoren wird, auf der Methode der Finiten Elemente basierend, hergeleitet. Die erweiterbare Implementierung orientiert sich stark an den Paradigmen der funktionalen Programmierung. Auf Grund der symbolischen Fähigkeiten von *Mathematica* können Fragestellungen symbolischer, komplexer und numerischer Natur gleichzeitig behandelt werden.

Abschließend werden mehrere Beispiele mit zunehmender Komplexität zur Verifikation der Algorithmen vorgestellt. Die Beispiele schließen solwohl das statische Verhalten als auch den Transport von kreisförmigen Blasen ein. Weiterhin werden eine Reihe von aufsteigenden Blasen aus verschiedenen Fluidbereichen sowie eine Ölblase die, durch eine Oberfläche bricht, vorgestellt. Als letztes Beispiel wird die Simulation eines Ink-Jet gezeigt. Die numerischen Experimente werden mit analytischen, numerischen und experimentellen Daten verglichen und eine gute Übereinstimmung wird festgestellt.

Wir haben gezeigt, daß mit verhältnismäßig einfachen Operatoren komplexe Phänomene wie Ink-Jet Simulation beschrieben werden können.

If you like this thesis and/or have questions do not hesitate to contact me under: *ruebenko*-AT-*imtek.de.*

The software is available on-line:

`http://www.imtek.uni-freiburg.de/simulation/mathematica/IMSweb/`

1 Introduction

Nel mezzo del cammin di nostra vita
mi ritrovai per una selva oscura
ché la diritta via era smarrita.
(Dante Alighieri)

1.1 Objectives of the Thesis **6**
1.2 Overview **6**
1.3 Major Results **7**
1.4 Document license **9**

SIMULATION is becoming well established. The application area of simulations range from quick feasibility checks to highly complicated models to gain fundamental understanding of complex natural phenomena. This thesis deals with the simulation of ink-jets, a fairly complicated and long winded process due to the demanding nature of the involved parameters. Small time and length scales as well as fluids with hard physical properties are to be dealt with. By taking a small detour - namely generality - we develop an ink-jet model, simulate and verify the model.

This thesis can be of value to you even if have no intention whatsoever to simulate ink-jets. We present techniques to deal with partial and ordinary differential equations in a very general manner. You will see how to computationally treat lumped systems such as electrical networks; no restriction implied here. The lumped systems can be from any physical kind such as from the mechanical, fluidic or the magnetic domain or a mixture there of.

Otherwise, a simple yet general approach to coupled non-linear partial differential equations is offered. Here, a way to automatically linearise partial differential equations is shown. The simplicity lies in the fact that only six so called operators are need to deal with partial differential equations up to second order time and and space derivatives, which allows for partial differential equations to be dealt with in a consistent way. These operators are implemented in an open source environment delivering both, arbitrary dimensional and symbolic finite element operators.

1.1 Objectives of the Thesis

The objective of this thesis was to develop an ink-jet printing simulation tool. A tool flexible enough to handle different print head geometries and ink material properties. Furthermore, the simulation should address the ejection of the ink as well as the flight of the droplet. The thesis actually goes one step beyond by including the landing of the ink droplet into the simulation.

1.2 Overview

The thesis is organised as follows. The first chapter gives a rough overview of the objectives and major results obtained in this thesis. In order to reach the objectives a very general approach to computer simulation has been taken. The engineering problem of simulating an ink-jet is set aside for the moment.

Instead of directly dealing with the ink-jet in Chapter 2 we present a general framework for the derivation of systems of ordinary differential equations. The chapter bases its generality on the observation that engineering problems commonly are described by nodes connecting elements; such as in electrical networks. The connectivity of the elements is represented in a graph. It is shown that a graph stemming from an engineering application can be converted to a system of ordinary equations by means of operators. Chapter 3 goes into the details of using lumped elements to build systems of equations. Here, in an exemplary manner, electrical network elements are used as lumped elements. It is, however, made clear that the concept is by no means restricted to lumped electrical elements. In Chapter 4 we introduce how to use partial differential equation operators in the same frame work, to, again, build systems of equations. Only a hand full of operators are necessary to model coupled non-linear partial differential equations.

Now, that all the tools are available, in Chapter 5 we model an ink-jet in the operator framework. Therefor the Navier-Stokes equation is automatically linearised and put into the partial differential equations frame work. For representing the interface between two fluids - the free surface - we use the well established level-set method.

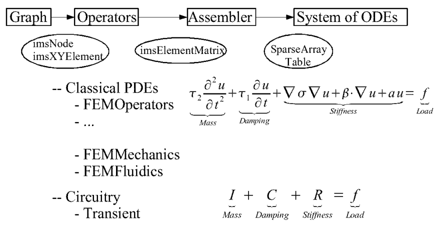
Up until now the operators are only specified in their interface, that is what is inserted into them and what is returned by them. Chapter 6 then describes the derivation of both, the lumped and partial differential equations operators. We show the mechanics that make the operators work. In this work we have chosen to use the finite element method for the partial differential equations operators. Also in this chapter you will find some information on boundary conditions as well as explanations of some challenges for the ink-jet model. Next, in Chapter 7 we give details of the actual implementation and show some sample sessions in the integrated engineering development environment. Finally, in Chapter 8 we extensively verify the program code with common free surface flow test cases. Last we show an ink-jet simulation. The thesis is concluded with an outlook of what could be done in the future to further enhance the obtained results. We also mention some experiments which could not be concluded successfully and discuss the possible reasons for the new challenges.

1.3 Major Results

In the opinion of the author the biggest success of the thesis is the fact that the software developed [4] is downloaded and used on a regular basis, among others by companies like Additive [1] and institutions like MIT [3] or CERN [2]. It seems useful to other people.

Integrated engineering development environment

We have developed and implemented a simple yet general and extendable modelling environment. The concept is based on entities called operators. Operators transform engineering components to systems of ordinary differential equations. The operator framework is suitable to coevally treat both lumped systems and possibly non-linear, coupled partial differential equations up to second order in space and time.
Chapter 2 introduces essentials of the transformation where Chapters 3 and 4 introduce lumped and partial differential equation operators respectively.


Graph
Operators
Assembler
System of ODEs
imsNode
imsXYElement
imsElementMatrix
SparseArray
Table
-- Classical PDEs
- FEMOperators
- ...
- FEMMechanics
- FEMFluidics
-- Circuitry
- Transient
$\underbrace{\tau_2 \frac{\partial^2 u}{\partial t^2}}_{Mass} + \underbrace{\tau_1 \frac{\partial u}{\partial t}}_{Damping} + \underbrace{\nabla \sigma \nabla u + \beta \cdot \nabla u + a u}_{Stiffness} = \underbrace{f}_{Load}$
$\underbrace{I}_{Mass} + \underbrace{C}_{Damping} + \underbrace{R}_{Stiffness} = \underbrace{f}_{Load}$

Different components of the work flow dealt with in this thesis.

Functional finite element & lumped operators

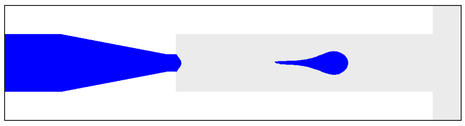
We have theoretically derived a generalised finite element master operator from “first principals”. Several specialised operators have been implemented. These arbitrary dimensional operators are, due to their implementation in *Mathematica*, suitable for numeric and complex imaginary computation or symbolic code generation. The operators handle arbitrary mixed element type meshes. Also lumped operators have been discretised to fit into the work-flow. Both derivation and discretisation are presented in Chapter 6.

$$\int_{\Omega^{(e_k)}} \coprod_{i=1}^{p} \kappa_i \otimes \mathcal{D}\left(N_i, \mathbf{r}, o_i\right)\, d\Omega^{(e_k)}$$

The integral representing the master finite element operator

Free surface fluid flow solver

As an application example we have developed a model for an ink-jet simulation. The model is based on the Navier-Stokes equation for the fluid flow and the Level-Set method for the free surface representation and is presented in Chapter 5. The results of the free surface flow solver are summarised in Chapter 8.


An ink-jet after ejecting.

1.4 Document license

References

[1] `http://www.additive-net.de/software/mathematica/packages`.

[2] `http://it-des.web.cern.ch/IT-DES/SIS/mathematics/mathematica_Addons.html`.

[3] `http://web.mit.edu/acs/www/numerical.html`.

[4] O. Rübenkönig and J. G. Korvink. Imtek mathematica supplement. `http://www.imtek.uni-freiburg.de/simulation/mathematica/IMSweb/`, 2002 – 2007.

2 Modelling with Operators

What makes it possible to study such a sequence of different applications is that mathematically they fit in the same framework. The theories are separate but parallel.
(Gilbert Strang)

2.1 Nodes, Elements & their Graphs **12**
2.2 Systems of Equations **13**
2.3 From Graphs to Systems **14**
2.4 Illustrations **15**

We present a general framework for the derivation of systems of ordinary differential equations, where special attention will be paid to three issues: First we observe that we would like to be able to deal with lumped models. Such models arise when physical properties are condensed and captured in an idealised element. An omnipresent task such as electrical circuit modelling makes use of lumped elements [2, 5]. At the same time we would like to be able to deal with partial differential equations.

To account for both situations, lumped models and partial differential equations, the derivation of the ordinary differential equations is based on the central observation that elements are connected to each other through nodes. To this end the notion of nodes, elements, and graphs is introduced. Graphs provide a convenient method to embrace nodes and elements. Therefore, graphs represent the starting point.

In a second step we introduce numerical systems of linear second order ordinary differential equations. Once we are able to establish such systems, standard analytical methods such as time integration or Eigenwert computation can be applied.

Now that we have introduced graphs and systems we show the general concept of operators in their role as a transforming entity. Operators transform elements and nodes of graphs to entries in systems of ordinary differential equations. Operators can be seen as the mediators between graphs and systems of equations. This mediation process is presented in two steps. First the process of applying operators to elements is discussed on a general level. Second, the assembly of matrices which define the system of ordinary differential equations is presented. The chapter will end with several examples, illuminating the theory.

2.1 Nodes, Elements & their Graphs

We start from the observation that engineering systems are constructed from "basic" components and these components are related to each other via some connectivity to be further specified. Along this connectivity we can transform components into parts of systems of linear ordinary differential equations. For concreteness, consider the components of an electrical circuit. Let us restrict ourselves, without loss of generality, to RLC networks. Such a network consists of two sorts of components:

1. elements
2. nodes.

The elements are connected to each other via nodes. Each element and the way it is connected contribute to a system of equations, more specifically, a system of second order linear ordinary differential equations.

The elements are not restricted to circuit elements but can be any element the engineer might need, for example lumped mechanical elements such as masses, dampers, and springs. To generalise even further, mesh elements for a finite discretisation method are of interest. Whatever the application, the elements and nodes form a graph.

Definitions: A graph [1] is a pair $G = (V, E)$ of sets such that for each element E we have the association $E \subseteq [V]^k$, where we denote by $[A]^k$ the set of all k-element subsets of A. A two node element would imply $k = 2$. A graph with node set V is said to be a graph on V. The node set of a graph G is referred to as $V(G)$, its edge set as $E(G)$. A node $n \in V(G)$ also written $n \in G$ is incident with an edge $e \in E(G)$ or $e \in G$ if $n \in e$; then e is an edge at n. Two nodes x, y are adjacent, or neighbours to each other if x, y is an edge of G. Two edges $e \neq f$ are adjacent, or neighbours if they have a node in common. If all nodes of G are pairwise adjacent then G is complete. The incidence matrix $B = (b_{ij})_{n\times m}$ of a graph $G = (V, E)$ with $V = \{n_1, .., n_n\}$ and $E = \{e_1, .., e_m\}$ is defined over the two element field $\mathbb{F}_2$ by

$$b_{ij} := \begin{cases} 1 & \text{if } n_i \in e_j \\ 0 & \text{otherwise.} \end{cases}$$

Then B and B^T define linear maps. $B : E(G) \mapsto V(G)$ and $B^T : V(G) \mapsto E(G)$. A directed graph assigns to every edge e an initial node i and a termination node t. The edge e is then said to be directed from i to t. A directed graph may have several edges between the same nodes x, y. Such edges are called multiple edges.

The incidence matrix B is never explicitly built. Instead, lists are established which store the actual incidents i_k for each element k. For each row i of b_{ij} only the values j are stored for which $b_{ij} = 1$.

Some graph elements may have more nodes than the so-called primary nodes which we have presented so far. Additional secondary nodes may be of interest for higher order (finite) elements.

2.2 Systems of Equations

Let

$$L = Su + D\dot{u} + M\ddot{u} \tag{2.1}$$

be a system of second order ordinary linear differential equations. S is the stiffness matrix, D the damping matrix, and M the mass matrix. u is the unknown and $\dot{u}$, $\ddot{u}$ are the first and second time derivatives respectively. L is the load vector, or a collection of load vectors, which is implied by the matrix notation for L. In matrix form the system of equations reads

$$\begin{pmatrix} l_1 \\ \vdots \\ l_n \end{pmatrix} = \begin{pmatrix} s_{11} & \cdots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{n1} & \cdots & s_{nn} \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} + \begin{pmatrix} d_{11} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{nn} \end{pmatrix} \begin{pmatrix} \dot{u}_1 \\ \vdots \\ \dot{u}_n \end{pmatrix}$$
$$+ \begin{pmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{nn} \end{pmatrix} \begin{pmatrix} \ddot{u}_1 \\ \vdots \\ \ddot{u}_n \end{pmatrix},$$

where for uncoupled systems the number of equations is $n = \max(n \in G)$. We call the coefficient matrices S, D, and M global matrices. Sometimes not all matrices are populated, in such cases the system may reduce to a first order or stationary system of equations.

For an extension to parametric systems see Moosmann [4] and for an extension to nonlinear systems see Lienemann [3]. In this work we show that for non-linear coupled partial differential equations a linear system is sufficient.

2.3 From Graphs to Systems

In this section we introduce the notion of operators and matrix assembly. Operators are mathematical entities which mediate between the graph and a system of equations. In essence an operators takes an element of the graph, applies its functionality and returns an element matrix. This so-called local element matrix is then built into the global system of equations, which is called the matrix assembly step.

2.3.1 Applying operators to graph elements

To arrive at the system of equations we take each element of the graph in turn and compute its contribution to the system of equations. This process is called application of operators $\mathcal{L}$ to graph elements e. An operator $\mathcal{L}$ is an entity that, given $e \in G$, computes the contribution of that one element to the system of equations.

$$\mathcal{L}(e) \rightarrow \{L^{(e)}, S^{(e)}, D^{(e)}, M^{(e)}\} \; \forall e \in G \tag{2.2}$$

Since the $X^{(e)}$ represent the contributions from each element they are called local element matrices. They indicate the local element load vector, element stiffness, and damping and mass matrix, respectively. Each local element matrix contributes to its equivalent global matrix.

Different applications may require different operators $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_n$. Each operator $\mathcal{L}_x$ may contribute to the load, stiffness, damping or mass matrices. A detailed discussion of the lumped operators can be found in Chapter 3, whereas the partial differential equation operators and their viable modelling are described in some detail in Chapter 4 . Once $\mathcal{L}(e)$ is computed the question remains as to how the local result is assembled into the global system of equations.

2.3.2 Assembly of local elements to global systems of equations

The last step in arriving at a system of equations is the matrix assembly step. Once an operator has computed the contribution of an element the result has to be put into the mass, damping, or stiffness matrix or the load vector. Depending on how the elements are connected with each other the element values are placed in the system matrices.

$$X_{n\in e_k} + X^{(e_k)}_{n\in e_k} \rightarrow X_{n\in e_k},$$

where X is any of the global system matrices, $n \in e_k$ is the connectivity of element k, and $X^{(e_k)}$ the local contribution from $\mathcal{L}(e_k)$.

2.4 Illustrations

The versatility of the presented framework is demonstrated in several examples.

2.4.1 Lumped element matrix assembly

Consider the lumped electrical circuit on the left of Figure 2.1. We have in red the vertices $V = \{n_1, n_2, n_3, n_4\}$ and in blue the elements $E = \{e_1, e_2, e_3, e_4, e_5\}$. The actual meaning of the elements is, for now, irrelevant. V and E make up a graph $G = (V, E)$. The uncoupled system of ordinary differential equations has $n = \max(n \in G) = 4$ degrees of freedom.

We apply the operators to each element. Here, the exemplary element e_2 is considered

$$\mathcal{L}(e_2) \rightarrow \{L^{(e_2)},\, S^{(e_2)},\, D^{(e_2)},\, M^{(e_2)}\},$$

where each of the local element matrices is a square matrix of size $m \times m$, with m the number of nodes which are incident to the element considered. With $\{n_2, n_4\} \in e_2 \in G$ it follows that $m = 2$. The local load vector contribution is $m \times 1$. The contribution of element e_2 to the global matrices thus is

$$\mathcal{L}(e_2) \rightarrow \left\{ \begin{pmatrix} l_2^{(e_2)} \\ l_4^{(e_2)} \end{pmatrix}, \begin{pmatrix} s_{22}^{(e_2)} & s_{24}^{(e_2)} \\ s_{42}^{(e_2)} & s_{44}^{(e_2)} \end{pmatrix}, \begin{pmatrix} d_{22}^{(e_2)} & d_{24}^{(e_2)} \\ d_{42}^{(e_2)} & d_{44}^{(e_2)} \end{pmatrix}, \begin{pmatrix} m_{22}^{(e_2)} & m_{24}^{(e_2)} \\ m_{42}^{(e_2)} & m_{44}^{(e_2)} \end{pmatrix} \right\}.$$

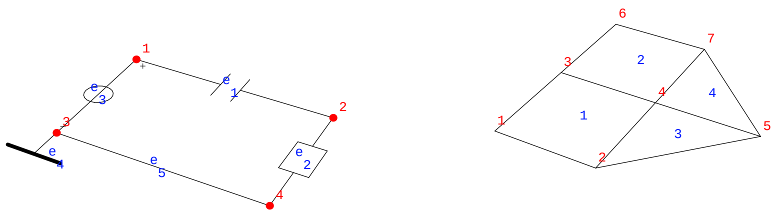


Figure 2.1: Left: A lumped electrical circuit. Depicted in red are the nodes $V = \{n_1, .., n_4\}$ and in blue the elements $E = \{e_1, .., e_5\}$. Right: A mesh consisting of four mesh elements. The node numbers are again depicted red and the element numbers blue.

In particular, we consider the way element e_2 contributes to the stiffness matrix

$$S + S^{(e_2)}_{n \in e_2} \rightarrow S,$$

in detail

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{pmatrix} + \begin{pmatrix} s^{(e_2)}_{22} & s^{(e_2)}_{24} \\ s^{(e_2)}_{42} & s^{(e_2)}_{44} \end{pmatrix} \rightarrow \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} + s^{(e_2)}_{22} & s_{23} & s_{24} + s^{(e_2)}_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} + s^{(e_2)}_{42} & s_{43} & s_{44} + s^{(e_2)}_{44} \end{pmatrix}.$$

In the same manner all local elements computed by the operator $\mathcal{L}$ are assembled into the global stiffness matrix. The values of the s_{xy} may be anything, typically they are zero when the matrix assembly process is started. The other global matrices are assembled in the same manner.

Remark: Without knowing the actual values of each local element we are nevertheless able to set up the global matrices according to the incidents given in the graph G.

2.4.2 Mesh element matrix assembly

The same procedure presented for lumped elements holds for mesh elements. On the right-hand side of Figure 2.1 we have a mesh with quadrilateral and triangular elements, in red the nodes $V = \{n_1, .., n_7\}$ and in blue the elements $E = \{e_1, .., e_4\}$. An uncoupled system of equations will thus consist of $n = \max(n \in G) = 7$ degrees of freedom. The elements' contribution for the quadrilateral elements $\{n_1, n_2, n_4, n_3\} \in e_1 \in G$ will be $m_{quad} = 4 \times 4$ and for the triangular elements $\{n_3, n_5, n_4\} \in e_3 \in G$ it will be $m_{tri} = 3 \times 3$.

If we introduce the short form g_{ij}^k, where k is the element e_k and i, j are incident to e_k, then the assembled global matrix will have contributions from the local elements in the following form

$$\begin{pmatrix}
g_{11}^1 & g_{12}^1 & g_{12}^1 & g_{13}^1 & g_{14}^1 & 0 & 0 & 0 \\
g_{21}^1 & g_{22}^1 + g_{22}^3 & g_{22}^1 + g_{22}^3 & g_{23}^1 & g_{24}^1 + g_{24}^3 & g_{25}^3 & 0 & 0 \\
g_{31}^1 & g_{32}^1 & g_{32}^1 & g_{33}^1 + g_{33}^2 & g_{34}^1 + g_{34}^2 & 0 & g_{36}^2 & g_{37}^2 \\
g_{41}^1 & g_{42}^1 + g_{42}^3 & g_{42}^1 + g_{42}^3 & g_{43}^1 + g_{43}^2 & g_{44}^1 + g_{44}^2 + g_{44}^3 + g_{44}^4 & g_{45}^3 + g_{45}^4 & g_{46}^2 & g_{47}^2 + g_{47}^4 \\
0 & g_{52}^3 & g_{52}^3 & 0 & g_{54}^3 + g_{54}^4 & g_{55}^3 + g_{55}^4 & 0 & g_{57}^4 \\
0 & 0 & 0 & g_{63}^2 & g_{64}^2 & 0 & g_{66}^2 & g_{67}^2 \\
0 & 0 & 0 & g_{73}^2 & g_{74}^2 + g_{74}^4 & g_{75}^4 & g_{76}^2 & g_{77}^2 + g_{77}^4
\end{pmatrix}$$

For the same graph the incident matrix is

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

To preserve the graph's direction the incidents are stored in mathematically positive orientation

$$i_1 : \{1, 2, 4, 3\},\ i_2 : \{3, 4, 7, 6\},\ i_3 : \{2, 5, 4\}, \text{ and } i_4 : \{4, 5, 7\}$$

i_k is the incident of element e_k. $i_{k,l}$ is the l^{th} part of i_k. These lists i_k may, for example, be used to traverse a mesh in a systematic manner or to establish a list of neighbouring elements or to find the elements connected to a node.

2.4.3 Mesh traversal

Some algorithms later in this work depend on a traversal of the mesh, or a graph, more generally speaking. Therefore, we present a method with which we will be able to do so. First, however, the way in which elements are connected to each other has to be found.

2.4.3.1 Connected elements

To find the connected elements we first define a list of empty sets of the length of the largest incident

$$s := \{\{\}_1, .., \{\}_{\max(\cup n)}\}.$$

Then the update function for one incident entry $i_{k,l}$ and counter $c \in \mathbb{I}$ returns a set with the old set for this incident s_i and the current counter

$$f(i_{k,l}, c) := s_{i_{k,l}} = \{s_{i_{k,l}}, c\}.$$

The update function is then applied to all incidents of one element

$$fe(i_k, c) := f(i_{k,1}, c), ..., f(i_{k,l}, c)$$

and in a last step the element update function is applied to all incidents

$$fc(i_1, .., i_k) := fe(i_1, 1), ..., fe(i_k, k).$$

For the example mesh above this results in the connected elements list e_c

$$e_c = \{\{1\}, \{1,3\}, \{1,2\}, \{1,2,3,4\}, \{3,4\}, \{2\}, \{2,4\}\}.$$

Here each entry in the row corresponds to one node and the elements connected to this one node. This means that for example node number 6 is only connected to the element 2.

2.4.3.2 Graph traversal

To traverse a graph we proceed as follows: First, all elements are marked as not visited

$$vi(e) := \text{False}.$$

The function `walk` finds the surrounding elements

$$\text{walk}(e_k) := \cup\,(e_c(\cup\,(i(e_k)))) \cap !vi(e)$$

where i are a list of incidences per element and e_k is the k-th element under inspection. Here $\cup$ is the union, which sorts and removes double entities in the list given to it. The function `step` marks elements as visited and walks the elements

$$\text{step}(e_k) := (vi(e_k) = \text{True}; \text{walk}(e_k);).$$

The function `step` can be nested while the element list is not an empty list $e \neq \{\}$.

Example: We traverse the above mesh and set all elements as not visited

$$vi(\{1,2,3,4\}) := \text{False}.$$

We wish to start from element 1 with

$$\text{step}(e_1) := (vi(e_1) = \text{True}; \text{walk}(e_1);$$

which leads to

$$\text{walk}(e_1) := (\cup e_c(\cup i(e_1))) \cap !vi(e)$$

with

$$\cup i(e_1) = i_1 = \{1,2,4,3\},$$

with e_c from above this leads to

$$\cup e_c(\{1,2,4,3\}),$$

$$\cup(\{1\},\{1,3\},\{1,2,3,4\},\{1,2\}) = \{1,2,3,4\}$$

and then to the result

$$\{1,2,3,4\} \cap !vi(e) = \{2,3,4\}.$$

In Figure 2.2 the traversal of a triangular mesh is displayed. Four randomly chosen elements mark the starting point for the traversal front which propagates in some 18 steps through the displayed mesh made up of roughly 1500 elements. A set of connected - possibly closed - elements could just as well have been chosen as starting elements.

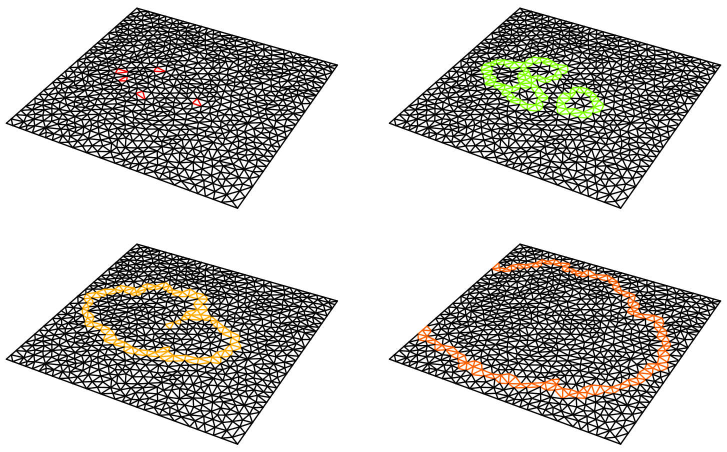


Figure 2.2: Graph traversal of a triangle mesh. The traversal is started at 4 randomly chosen elements from which the traversal front propagates. Three further time steps are displayed. The mesh consists of roughly 1500 elements.

References

[1] Reinhard Diestel. *Graph Theory*. Springer-Verlag Heidelberg, New-York, third edition, 2005.

[2] Stefan Jahn and Michael Margraf. qucs. `http://qucs.sourceforge.net/`.

[3] Jan Lienemann. *Complexity Reduction Techniques for Advanced MEMS Actuators Simulation*. PhD thesis, University Freiburg, 2006.

[4] E. B. Moosmann, C.and Rudnyi, A. Greiner, and J. G.and Hornung M. Korvink. Parameter preserving model order reduction of a flow meter. In *Technical Proceedings of the 2005 Nanotechnology Conference and Trade Show*, volume 3, pages 684 – 687, 2005.

[5] L. Nagel and D. Pederson. Spice. `http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/`, 1975.

3 Modelling with Lumped Operators

9 weniger 10 - das geht nicht.
Da müssen wir uns eins borgen.
(Alte Schulweisheit)

3.1 Lagrangian Dynamics **22**
3.2 Modified Nodal Approach **25**
3.3 Lagrangian Modified Nodal Approach **25**

LUMPED simulation deals with the simulation of elements where the elements' physical properties could be condensed into mostly one continuous equation. For example, Ohm's[1] law $U = R\,I$ represents such an equation were the material property resistance R relates the current I to the voltage U. Such relations can be found in numerous engineering areas: thermal, mechanical, and fluidic elements are not the only known items.

[1]Georg Simon Ohm (1789-1854) published a pamphlet "Die galvanische Kette mathematisch bearbeitet" about the theory and applications of electrical current, where he introduced "Ohm's law".

Lumped operators are the mathematical operators that transform a continuous lumped equation into a discrete set of equations suitable for assembly into a system of equations such as presented in Equation 2.1. In the following we show the derivation of the lumped operators for circuit elements based on Lagrangian dynamics. First a short revision of Lagrangian dynamics for circuit elements is given. Since Lagrangian dynamics has some shortcomings for a computer implementation, in the next section we introduce the Modified Nodal Approach which will be shown in the final section to circumvent the previous shortcomings.

3.1 Lagrangian Dynamics

In Lagrangian dynamics [3] two formulations for circuit models are common: the charge formulation and the flux linkage formulation. The charge formulation is based on finding closed loops for the analysis based on Kirchhoff's[2] voltage rule. Since in our system approach we focus on nodal values, the flux linkage formulation seems more appropriate. The flux formulation is based on Kirchhoff's current law which states that the sum of the currents entering any node must be zero. The requirements for the system must then be of the form

$$\frac{d\lambda_k}{dt} = e_k, \tag{3.1}$$

where λ_k is the k^{th} generalised coordinate. λ is the unacquainted flux linkage in Weber[3] and e the voltage in Volt.

Lagrange's[4] equation is given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\lambda}_k}\right) - \frac{\partial L}{\partial \lambda_k} = \mathcal{I}_k.$$

The electrical circuit Lagrangian function L, usually simply called the Lagrangian, for flux linkage variables is defined as

$$L = W_e^* - W_m,$$

where W_e^* is the electric co-energy and W_m the magnetic energy. W_e^* is populated through capacitive elements and W_m through inductances. $\mathcal{I}_k$ is the

[2]Gustav Robert Kirchhoff (1824-1887) formulated the circuit laws while he was a student in 1845.
[3]Wilhelm Eduard Weber (1804-1891) in 1831, on the recommendation of Gauß, he was called to Göttingen as professor of physics, although but twenty-seven years of age.
[4]Joseph-Louis, Comte de Lagrange (1736-1813) is regarded as the greatest mathematician of the 18th century after Leonhard Euler. Lagrange made, among other things, important contributions to mechanics. By the age of 20 he held a chair for geometry.

contribution of generalised non-conservative currents

$$\delta\mathcal{W}^{nc} = \sum_{i=1}^{N} \mathcal{I}_k \, \delta\lambda_k.$$

The $\delta\mathcal{W}^{nc}$ part comprises current sources and resistances. The following table composes the primary idealised elements in the flux linkage formulation.

W_e^*	W_m	$\delta\mathcal{W}^{nc}$
$\frac{C\dot{\lambda}^2}{2}$	$\frac{\lambda^2}{2L}$	$I(t)\,\delta\lambda$
		$-\frac{1}{R}\dot{\lambda}\,\delta\lambda$

Example: We consider the circuit in Figure 3.1 and derive the system of equations via Lagrange's formulation:

1. Generalised coordinates

$$\lambda_k : \lambda_1 \; \lambda_2 \; \lambda_3 \; \lambda_4$$

here $\lambda_1 = \lambda_5 = \lambda_6$. Strictly speaking also λ_1 is not neccessary since the potential is equal to 0.

2. Lagrangian

$$W_e^* = \frac{1}{2}C_1\left(\dot{\lambda}_3 - \dot{\lambda}_2\right)^2 + \frac{1}{2}C_2\left(\dot{\lambda}_4 - \dot{\lambda}_3\right)^2$$

$$W_m = \frac{\lambda_2^2 - \lambda_1^2}{2L}$$

3. Generalised currents

$$\delta\mathcal{W}^{nc} = -\frac{1}{R}(\dot{\lambda}_4 - \dot{\lambda}_1)\,(\delta\lambda_4 - \delta\lambda_1) + I(\delta\lambda_3 - \delta\lambda_1) = \sum_{k=1}^{2} \mathcal{I}_k \, \delta\lambda_k$$

with $\dot{\lambda}_1 = \dot{\lambda}_5$.

4. Lagrange's equation applied then results in

$$\begin{pmatrix} -I(t) \\ 0 \\ I(t) \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{L} & -\frac{1}{L} & 0 & 0 \\ -\frac{1}{L} & \frac{1}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{pmatrix} +$$

$$\begin{pmatrix} \frac{1}{R} & 0 & 0 & -\frac{1}{R} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{R} & 0 & 0 & \frac{1}{R} \end{pmatrix} \begin{pmatrix} \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \\ \dot{\lambda}_4 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & C_1 & -C_1 & 0 \\ 0 & -C_1 & (C_1 + C_2) & -C_2 \\ 0 & 0 & -C_2 & C_2 \end{pmatrix} \begin{pmatrix} \ddot{\lambda}_1 \\ \ddot{\lambda}_2 \\ \ddot{\lambda}_3 \\ \ddot{\lambda}_4 \end{pmatrix}$$

Several comments concerning this example are appropriate: In the classical formulation it is not necessary to include λ_1. In this work we, however, include every node in the schematic. From a computer implementation point of view it is far easier and more general to later deal with them in an appropriate manner than to previously decide which nodes to include and which not. Furthermore, not removing nodes is very instructive for the construction of lumped operators. We thus see by inspection that an element connecting nodes i and j contributes to the appropriate global matrix at positions $\{\{i,i\},\{i,j\},\{j,i\},\{j,j\}$. For example, the capacitance C_1 is to be found in four positions in the mass matrix[5]; solely specified by the identification numbers of the embracing nodes λ_2 and λ_3. It is then possible to assemble the system of equations automatically.

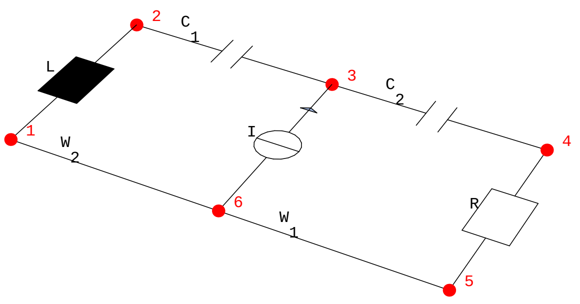


Figure 3.1: A small electrical circuit with inductance L, current source I, two capacitances C_1 and C_2, a resistance R, and ground G. Wires W_1 and W_2 are also shown.

Remark: The unknown variable is λ. However, we are much more interested in the voltage e_k which can be computed by Equation 3.1

Several open questions remain:

1. How to automatically deal with the nodes which are on the same potential as λ_1 as has been done in the example above?
2. How to build in voltage sources without transforming them to current sources?
3. How to deal with the wires between nodes λ_1, λ_5 and λ_6?

[5]We keep the matrices' names as introduced in Section 2.2. These names stem from the domain of mechanics and are instructive. For example, in a flux linkage formulation the resistive parts make up the damping of the system of equations.

3.2 Modified Nodal Approach

The questions posed in the previous section were essentially the same as in the nodal analysis prior to 1975 when the Modified Nodal Approach [1] paper was published. The nodal approach (and as a matter of fact the modified nodal approach too) differs from the Lagrangian method in the fact that the voltage e_k is the unknown variable and not λ_k as in the Lagrangian method. We can use the modified nodal approach to circumvent the above-mentioned shortcomings.

In the modified nodal approach a system of equations of the form

$$Y \cdot U = J$$

is formulated were Y is the admittance matrix, U represents the sought nodal voltages, and J the current source vector. The modified nodal approach introduces branch currents as additional variables and the constitutive relations as additional equations to the form

$$\begin{bmatrix} Y_R & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} U \\ I \end{bmatrix} = \begin{bmatrix} J \\ E \end{bmatrix},$$

where Y_R is the reduced form of the nodal matrix excluding contributions due to voltage sources, current controlling elements, and the like. B contains the partial derivative of the Kirchhoff current equations with respect to the additional current variables. This implies a value of ± 1 for the introduced branch relations. The branch constitutive relations, differentiated with respect to the unknown vector, are represented in matrices C and D. This is shown further down with more detail.

3.3 Lagrangian Modified Nodal Approach

We now adapt the modified nodal approach to the Lagrangian method. Since we use a flux linkage formulation we can say that the resistive components belong to the damping matrix, the inductive components to the stiffness matrix, and the capacitive components to the mass matrix. The exact discretisation can be found in Chapter 6. The method is best illustrated by an example.

Example: The following system of equations is the result of a Lagrangian modified modal approach applied to the circuit in Figure 3.1 and will be explained below

$$\begin{pmatrix} 0 \\ 0 \\ i \\ 0 \\ 0 \\ -i \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{L} & -\frac{1}{L} & 0 & \cdots & 0 \\ -\frac{1}{L} & \frac{1}{L} & 0 & \cdots & \\ 0 & 0 & 0 & \cdots & \\ \vdots & \vdots & \vdots & \ddots & \\ & & & & \\ 0 & & & & 0 \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \\ \lambda_7 \\ \lambda_8 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{R} & -\frac{1}{R} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{R} & \frac{1}{R} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot$$

$$\begin{pmatrix} \dot\lambda_1 \\ \dot\lambda_2 \\ \dot\lambda_3 \\ \dot\lambda_4 \\ \dot\lambda_5 \\ \dot\lambda_6 \\ \dot\lambda_7 \\ \dot\lambda_8 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & C_1 & -C_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -C_1 & C_1 + C_2 & -C_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -C_2 & C_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \ddot\lambda_1 \\ \ddot\lambda_2 \\ \ddot\lambda_3 \\ \ddot\lambda_4 \\ \ddot\lambda_5 \\ \ddot\lambda_6 \\ \ddot\lambda_7 \\ \ddot\lambda_8 \end{pmatrix}.$$

For now the ground is ignored and not present in the system of equations. Equations 1-6 are added like in a Lagrangian method. The current source is inserted in the load at nodes 3 and 6.
Only the voltage source enlarges the system of equations, hence the submatrices C and B in the modified nodal approach. Wires are handled like voltage sources, only that the voltage U is 0. We see that the system has been enlarged by equations for λ_7 and λ_8. In essence the added equations are $\lambda_5 - \lambda_6 = 0$ and $-\lambda_1 + \lambda_6 = 0$. Similarly, if a voltage source were located between nodes 5 and 6 then the equations were $\lambda_5 - \lambda_6 = U$, where U is the voltage of the source. Please note that in this case $B^T = C$.
Adding a ground imposes a Dirichlet boundary condition of value 0 and offsets are allowed. The mechanism which describes how to insert the boundary condition is discussed in Chapter 7.

The element matrices of the different circuit elements are called stamps and are presented in Chapter 6.

Lagrange's Method was originally derived for mechanical systems and thus a system which applies to mechanics is possible within the same framework. The extensions to the fluid domain are of interest and can be found elsewhere [2]. Furthermore, the combination of the different domains is possible. An extension to the micro electro-mechanics domain with its comb-drives and other devices might be promising.

References

[1] Chung-Wen Ho, Albert E. Ruehli, and Pierce A. Brennan. The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems*, CAS-22, No. 6:504 – 509, 1975.

[2] Timo Lindemann. *Droplet Generation From the Nanoliter to the Femtoliter Range*. PhD thesis, University Freiburg, 2006.

[3] James H. Jr. Williams. *Fundamentals of Applied Dynamics*. John Wiley & Sons, Inc., 1996.

4 Modelling with Partial Differential Equation Operators

Das Gegenwärtige ist begrenzt.
Das Mögliche unermesslich.
(Unknown)

4.1 Building Blocks 30
4.2 Combining Operators to Model Partial Differential Equations 34

SINCE the early days of computerisation, scientist from all fields have been making use of computers to automate complex computational tasks. Obviously, modelling nature is the fundamental challenge, the primary goal being to promote the understanding of nature and its innermost working. Independent, but hugely promoted by the computer revolution, different methodologies for projecting observed natural phenomena into the domain of physics evolved - the common tongue being mathematics. In this work we use differential equations, which, however, are not solely amenable for modelling natural phenomena. We first provide building blocks for partial differential equations which then are shown to be suitable for treating coupled and non-linear systems of equations.

4.1 Building Blocks

We start by looking at mathematical entities we call differential operators. In the broadest sense a differential operator evokes function differentiation. In a partial differential equations (PDE) context, differential operators differentiate an unknown, sought after function u. In order to be able to treat a wide range of partial differential equations we are about to discuss a general set of differential operators. In other words, we learn to view partial differential equations not as a single equation but as equations composed of basic building blocks. As a prerequisite we look at the Nabla operator.

Nabla operator

The Nabla operator[1] ∇ is a pseudo vector and indicates the notion of differentiation. The operator, however, is completely independent from coordinates and dimensions; those are defined solely by the function to be differentiated. In n-dimensional space $\mathbb{R}^n$

$$\nabla\Box = \begin{pmatrix} \frac{\partial\Box}{\partial x_1} \\ \vdots \\ \frac{\partial\Box}{\partial x_n} \end{pmatrix}$$

returns all partial derivatives of a scalar function $\Box$. This is known as the gradient. Here $\nabla : \mathbb{R} \mapsto \mathbb{R}^n$. The differentiation property acts on the character to its right, while the vector property is applied as a normal vector. ∇ applied to a vector field

$$\nabla \cdot \underline{\Box} = \left(\sum_i^n \frac{\partial \Box_{x_i}}{\partial x_i} \right)$$

where in this case $\nabla : \mathbb{R}^n \mapsto \mathbb{R}$.

In the following, we consider scalar unknown fields denoted by u.

4.1.1 Diffusion operator

The diffusion operator models 2^{nd} order derivatives in space as for example in the Laplace[2] equation

$$\nabla^2 u = 0$$

[1]The operator was introduced by Hamilton, who used the rotated form $\triangleleft$ and it was P. G. Tait who established the form known today. The symbol ∇ is possibly derived from a Hebrew string instrument close to a harp which had a similar form. The resemblance of the ∇ operator to an Assyrian harp made R. Smith suggest the $\epsilon\lambda\lambda\eta\nu\iota\kappa\eta\,\gamma\lambda\omega\pi\alpha$ (old Greek) name of nabla.

[2]Pierre-Simon, Marquis de Laplace (1749-1827) translated the geometric studies of mechanics used by Isaac Newton to one based on calculus. He also discovered the Laplace equation.

The equation describes a potential distribution in a medium and consists of two parts. The diffusion operator $\nabla^2 u$ and a right-hand side term, which is 0. In the form presented, the equation holds for isotropic media. To extend the equation in order to be able to handle anisotropic media we rewrite the equation in the following form

$$\nabla^T \cdot (-\sigma \nabla u) = 0,$$

where σ is a rank-2 tensor and can be a function of the same coordinates as the ∇ operator. The dimension of σ is a function of the dimension of ∇. The following holds in $\mathbb{R}^n$

$$\begin{aligned} \dim\left(\nabla^T\right) &= \{1, n\} \\ \dim(\sigma) &= \{n, n\} \\ \dim(\nabla) &= \{n, 1\}. \end{aligned}$$

Different settings for σ model different, possibly anisotropic, media. By setting

$$\sigma_{ij} = \delta_{ij}$$

and noting that δ_{ij} is the Kronecker[3] delta we can again model isotropic media. The diffusion operator in Equation 4.1 is the single most important operator.

$$\nabla^T \cdot (-\sigma \nabla u) \tag{4.1}$$

4.1.2 Load operator

The next step is to model Poisson's equation

$$\nabla^2 u = L(f)$$

for which we need to incorporate a right-hand side different from 0 . This is achieved with the load operator.

$$L \tag{4.2}$$

The most notable difference of the load operator, besides its functionality, is the fact that this operator does not involve the unknown function u. L computes the load f to the equation. f can, nevertheless, be a function of the coordinates of the ∇ operator. The following holds in $\mathbb{R}^n$

$$\dim(f) = 1$$

[3]Leopold Kronecker (1823 - 1891) was a German mathematician. The Kronecker delta is defined as $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

4.1.3 Reaction operator

To model Helmholtz's equation

$$\nabla^2 u + au = f$$

we additionally need a so-called reaction operator. The reaction operator

$$au \tag{4.3}$$

models equations with 0^{th} order derivative. a is a rank-0 tensor (scalar) and in the physical context the operator can be interpreted as local change in u due to sources, sinks and reactions. a can be a function of the coordinates of the ∇ operator. The following holds in $\mathbb{R}^n$

$$\dim(a) = 1.$$

Sometimes the reaction operator is called absorption operator.

4.1.4 Convection operator

Now that we have 2^{nd} and 0^{th} order spatial operators we accommodate 1^{st} order spatial derivatives. We introduce the convection operator

$$\beta \cdot \nabla u. \tag{4.4}$$

In the physical context β is a rank-1 tensor and can physically be interpreted as the velocity with which a convection takes place. β can be a function of the coordinates of ∇. The following holds in $\mathbb{R}^n$

$$\begin{aligned} \dim(\beta) &= \{1, n\} \\ \dim(\nabla) &= \{n, 1\}. \end{aligned}$$

4.1.5 Conservative flux convection operator

The following operator does not have much of a physical meaning. The main functionality of this operator is in the linearisation process of non-linear partial differential equations (see Section 4.2.4). Terms involving the conservative flux convection operator will appear at a later stage.

$$\nabla^T \cdot (-\alpha u) \tag{4.5}$$

The following holds

$$\begin{aligned}\dim\left(\nabla^T\right) &= \{1, n\} \\ \dim(\alpha) &= \{n, 1\}.\end{aligned}$$

4.1.6 Load derivative operator

The following operator does not have much of a physical meaning. The main functionality of this operator is in the linearisation process of non-linear partial differential equations (see Section 4.2.4). Terms involving the load derivative operator will appear at a later stage.

$$\nabla^T \cdot (\gamma)$$

The following holds

$$\begin{aligned}\dim\left(\nabla^T\right) &= \{1, n\} \\ \dim(\gamma) &= \{n, 1\}.\end{aligned}$$

4.1.7 1^{st} order time derivative

To model transient behaviour of the first kind we introduce the 1^{st} order transient operator

$$\tau_1 \frac{\partial}{\partial t} u. \tag{4.6}$$

This operator models the first time derivative of function u. τ_1 is a rank-0 tensor (scalar) and in the physical context τ_1 is a damping constant. τ_1 can be a function of the coordinates of ∇. The following holds in $\mathbb{R}^n$

$$\dim(\tau_1) = 1.$$

4.1.8 2^{nd} order Time Derivative

To model the wave equation

$$\frac{\partial^2}{\partial t^2} u - c\nabla^2 u = 0$$

we introduce the 2^{nd} order transient operator:

$$\tau_2 \frac{\partial^2}{\partial t^2} u. \tag{4.7}$$

τ_2 is a rank-0 tensor (scalar) and in the physical context τ_2 is an inertia constant. τ_2 can be a function of the coordinates of ∇. The following holds in $\mathbb{R}^n$

$$\dim(\tau_2) = 1.$$

4.1.9 Operators: A different viewpoint

To summarise: We have n-dimensional scalar valued operators from zeroth order to second order in space and time. Where each differential operator has a so-called input function which specifies the operator's physical behaviour. The input function may depend on the coordinates defined by ∇. Physical behaviour can thus be switched on and off in different regions of the simulation domain Ω. This topic will be broadened subsequently.

4.2 Combining Operators to Model Partial Differential Equations

Now that we have introduced the building blocks we begin to combine these differential operators.

4.2.1 Coefficient Form

We postulate the partial differential equation 4.8 to be fundamental [2, 1]:

$$\tau_2 \frac{\partial^2}{\partial t^2} u + \tau_1 \frac{\partial}{\partial t} u + \nabla^T \cdot (-\sigma \nabla u - \alpha u + \gamma) + \beta \cdot \nabla u + au = f \tag{4.8}$$

Even though Equation 4.8 has limited physical meaning as such, it is possible to map a wide range of problems from the physical domain onto the mathematical domain, which makes for a great deal of the equation's elegance. Essentially we view Equation 4.8 as the sum of the operators defined in Equations 4.1 to 4.7. The presence of all operators is not mandatory. With the combination of the operators we are able to model a wide range of scalar-valued linear partial differential equations with at most 2^{nd} order time derivatives and at most 2^{nd} order space derivatives in $\mathbb{R}^n$. For concreteness, Table 4.1 of selected partial differential equations and the necessary input functions for the operators is presented.

4.2.2 Transient convection-diffusion example

Consider the following problem: In a piece of bulk silicon we have an embedded heater device (see Figure 4.1). To the right and left thereof two temperature

Table 4.1: Classical partial differential equations and the necessary input functions for the differential operators. The different equations are presented in different space dimensions n. For the equations additional problem/material dependent parameters (a, b, c, ρ, k, V) have to be specified which may be dependent on the coordinates in $\mathbb{R}^n$. Here $\mathbf{I}$ is the identity matrix in n dimensions.

Name	Partial Differential Equation	n	τ_2	τ_1	σ	a	f
Telegraph	$\tau_2 \frac{\partial^2}{\partial t^2} u + \tau_1 \frac{\partial}{\partial t} u - \nabla^T \sigma \nabla u + a\, u = 0$	1	1	$a + b$	$c\,\mathbf{I}$	$a * b$	0
Heat	$-\nabla^T \sigma \nabla u = \rho(x, y)$	2	0	0	$c\,\mathbf{I}$	0	ρ
Schrödinger	$\tau_1 \frac{\partial}{\partial t} u - \nabla^T \sigma \nabla u + a\, u = 0$	2	0	$\imath\hbar$	$-\frac{\hbar^2}{2m}\mathbf{I}$	$-V$	0
Wave	$\tau_2 \frac{\partial^2}{\partial t^2} u - \nabla^T \sigma \nabla u = 0$	3	1	0	$c\,\mathbf{I}$	0	0
Helmholtz	$-\nabla^T \sigma \nabla u + a\, u = 0$	2	0	0	$\mathbf{I}$	k^2	0
Klein-Gorden	$\tau_1 \frac{\partial}{\partial t} u - \nabla^T \sigma \nabla u + a\, u = 0$	2	0	$\frac{1}{c^2}$	$\mathbf{I}$	k^2	0

sensors are embedded. If the heater is switched on, both sensors will measure the same temperature. Once a flow field is flowing over the silicon device, the temperature distribution will shift according to the direction and magnitude of the flow field. From the temperature difference between the two sensors, it is possible to deduce the speed of the flow field. This is the principle of an anemometer build by Ernst [3, 4].

The system can be modelled by the following operators:

$$\tau_1 \frac{\partial}{\partial t} u - \nabla^T \sigma \nabla u + \beta \cdot \nabla u = f,$$

where u is the temperature distribution. We subdivide the simulation domain Ω into 4 sub domains where $\cup \Omega_{i=1..4} = \Omega$. Each of the operators is active in all regions, which, however, can mean that the contributing is zero. The input functions in this context are the density times specific heat τ_1 , the heat conduction σ , the velocity times density times specific heat β , and the heat source f. The input functions themselves can be functions of space, so they switch on and off different parameters. The following table summarises the values of the input functions:

	$\tau_1 = \rho\, c$	σ	$\beta = v\, \tau$	f
	10^{-6} [Kg/m s K]	10^{6} [Kg m/s^2 K]	10^{3} [Kg/s^2 K]	10^{-6} [Kg/s^2 m]
Ω_1/ bulk Si	$2330 * 700$	$148 * \mathbf{I}$	$\begin{pmatrix} 0 & 0 \end{pmatrix}$	0
Ω_2/ heater Si	$2330 * 700$	$148 * \mathbf{I}$	$\begin{pmatrix} 0 & 0 \end{pmatrix}$	10^{10}
Ω_3/ free cut	$1000 * 1.29$	$0.024 * \mathbf{I}$	$\begin{pmatrix} 0 & 0 \end{pmatrix}$	0
Ω_4/ flow zone	$1000 * 1.29$	$0.024 * \mathbf{I}$	$\begin{pmatrix} 1.29 * p & 0 \end{pmatrix}$	0

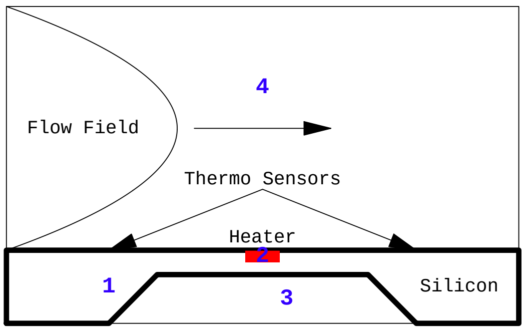


Figure 4.1: The anemometer sketch depicts a piece of bulk silicon (region 1) with an embedded heater device (2). Under the silicon device we have a free cut area (3) and above the silicon the flow field area (4). Once the heater is switched on and a flow passes the heater a temperature difference can be measured at the thermal sensors. Based on the magnitude of the temperature difference and material data the flow speed can be estimated.

We set $p = \left((y-a-b)^2/b^2 - 1\right)$. Here a and b are parameters for the flow profile. In Figure 4.2 we present the solution at different time steps.

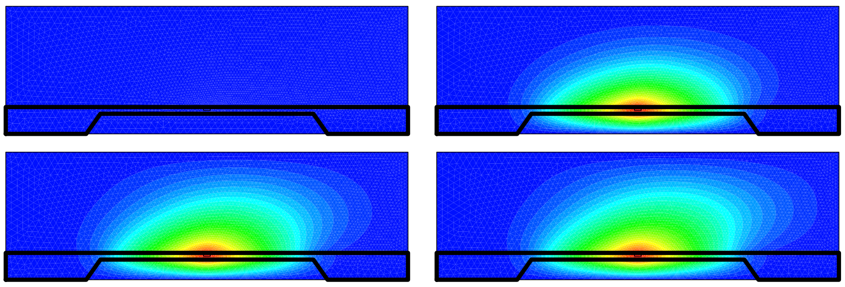


Figure 4.2: Anemometer solution at four different time steps.

Remark: A plausibility check can be performed by inserting the input functions into the partial differential equation and checking that the units are in agreement.

4.2.3 Coupled systems of partial differential equations

For the next level of complexity we would like to model coupled systems of equations. Now, the sought after function u is vector-valued. We present a technique to model systems of equations with the previously defined scalar-valued differential operators. No new operators need to be introduced. The procedure is best described by several examples

Example: The following system of two partial differential equations is given and we would like to find the input functions to the differential operators that represent the same system of equations

$$\begin{aligned} 4\frac{\partial}{\partial y^2}v(x,y) + \frac{\partial}{\partial x^2}u(x,y) = &\quad 0 \\ 9\frac{\partial}{\partial y^2}u(x,y) + \frac{\partial}{\partial x^2}v(x,y) = &\quad 0. \end{aligned}$$

First we write in matrix notation and thus rearrange the equation to

$$\begin{pmatrix} \frac{\partial}{\partial x^2} & 4\frac{\partial}{\partial y^2} \\ 9\frac{\partial}{\partial y^2} & \frac{\partial}{\partial x^2} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Next we subdivide the matrix

$$\left(\begin{array}{c|c} a_{11} & a_{12} \\ \hline a_{21} & a_{22} \end{array}\right) \begin{pmatrix} u \\ v \end{pmatrix} = \left(\begin{array}{c|c} \nabla\cdot(\sigma_{11}\nabla) & \nabla\cdot(\sigma_{12}\nabla) \\ \hline \nabla\cdot(\sigma_{21}\nabla) & \nabla\cdot(\sigma_{22}\nabla) \end{array}\right) \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Now we consider each part of the matrix separately. Which input functions are sought depends on the order of differentiation present in the system of equations. In this case we have second order spacial derivatives. This implies we seek the input functions σ of the diffusion operator. The number of input functions needed to be specified is the number of unknowns squared. Here we have 2 unknowns, namely u and v. So we will need 4 σ input functions. One for each quadrant. For the matrix entry a_{11} we ask the question: What must the input function σ_{11} look like to model $\partial/\partial x^2$. This is achieved by selecting $\sigma_{11} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$. In this same manner all a_{ij} are handled. We obtain the 4 input functions as

$$\left(\begin{array}{c|c} \sigma_{11} & \sigma_{12} \\ \hline \sigma_{21} & \sigma_{22} \end{array}\right) = \left(\begin{array}{c|c} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 4 \end{pmatrix} \\ \hline \begin{pmatrix} 0 & 0 \\ 0 & 9 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}\right).$$

A generalised way of writing the coefficient matrix of the diffusion operator's input function can be derived from the special case where we have a vector-valued unknown function composed of $s = 2$ scalar-valued functions (e.g. u and v) in $\mathbb{R}^{n=2}$ (e.g. x and y)

$$\left(\begin{array}{c|c} \sigma_{11} & \sigma_{12} \\ \hline \sigma_{21} & \sigma_{22} \end{array}\right) = \left(\begin{array}{c|c} \begin{pmatrix} \sigma_{1111} & \sigma_{1112} \\ \sigma_{1121} & \sigma_{1122} \end{pmatrix} & \begin{pmatrix} \sigma_{1211} & \sigma_{1212} \\ \sigma_{1221} & \sigma_{1222} \end{pmatrix} \\ \hline \begin{pmatrix} \sigma_{2111} & \sigma_{2112} \\ \sigma_{2121} & \sigma_{2122} \end{pmatrix} & \begin{pmatrix} \sigma_{2211} & \sigma_{2212} \\ \sigma_{2221} & \sigma_{2222} \end{pmatrix} \end{array}\right) = \sigma_{ijkl}$$

where $i, j = 1, .., s$ and $k, l = 1, .., n$.

The input for other operators can be found in a similar manner. For the convection operator we have

$$\beta_{ijk}$$

where $i, j = 1, .., s$ and $k = 1, .., n$. For two unknowns this means

$$\left(\frac{\beta_{11} \mid \beta_{12}}{\beta_{21} \mid \beta_{22}}\right) = \left(\frac{\left(\begin{array}{cc} \beta_{111} & \beta_{112} \end{array}\right) \mid \left(\begin{array}{cc} \beta_{121} & \beta_{122} \end{array}\right)}{\left(\begin{array}{cc} \beta_{211} & \beta_{212} \end{array}\right) \mid \left(\begin{array}{cc} \beta_{221} & \beta_{22} \end{array}\right)}\right) = \beta_{ijk}.$$

For the second and first transient operator and the reaction operator we have

$$\psi_{ij},$$

where $i, j = 1, .., s$. Here we use ψ to represent τ_1, τ_2 and a. For two unknowns this means

$$\left(\frac{\psi_{11} \mid \psi_{12}}{\psi_{21} \mid \psi_{22}}\right).$$

For the load operator we have

$$f_i,$$

where $i = 1, .., s$. For two unknowns this means

$$\left(\frac{f_1}{f_2}\right).$$

The next example shows how to model the $\times$ (cross or curl) operator with the basic operators.

Example: In order to model the Coriolis force the $\times$ operator is of importance. Here we illustrate how to use the basic operators to model the curl operator. Note that

$$(2\omega \times \mathbf{v}) = 2\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \times \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = 2\begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}.$$

The input functions for the 9 convection operators are

$$2\left(\begin{array}{ccc} \left(\begin{array}{ccc} 0 & 0 & 0 \end{array}\right) & \left(\begin{array}{ccc} 0 & 0 & -1 \end{array}\right) & \left(\begin{array}{ccc} 0 & 1 & 0 \end{array}\right) \\ \left(\begin{array}{ccc} 0 & 0 & 1 \end{array}\right) & \left(\begin{array}{ccc} 0 & 0 & 0 \end{array}\right) & \left(\begin{array}{ccc} -1 & 0 & 0 \end{array}\right) \\ \left(\begin{array}{ccc} 0 & -1 & 0 \end{array}\right) & \left(\begin{array}{ccc} 1 & 0 & 0 \end{array}\right) & \left(\begin{array}{ccc} 0 & 0 & 0 \end{array}\right) \end{array}\right).$$

Example: Maxwell's equation

$$\nabla \times \frac{1}{\mu_0 \mu_r}(\nabla \times A)$$

can also be modelled. We view this as a system of equations and write

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} Ax \\ Ay \\ Az \end{pmatrix},$$

with the diffusion operator coefficients

$$\begin{pmatrix}
\begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{\mu_0\mu_r} & 0 \\ 0 & 0 & \frac{1}{\mu_0\mu_r} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{\mu_0\mu_r} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -\frac{1}{\mu_0\mu_r} & 0 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & -\frac{1}{\mu_0\mu_r} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} \frac{1}{\mu_0\mu_r} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\mu_0\mu_r} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -\frac{1}{\mu_0\mu_r} & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & -\frac{1}{\mu_0\mu_r} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\mu_0\mu_r} \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} \frac{1}{\mu_0\mu_r} & 0 & 0 \\ 0 & \frac{1}{\mu_0\mu_r} & 0 \\ 0 & 0 & 0 \end{pmatrix}
\end{pmatrix}.$$

Without adding any new operators we have expanded the basic operator's functionality tremendously by viewing coupled partial differential equations from the right angle.

4.2.4 Non-linear systems & the General Form

Now that we have a mechanism for treating coupled PDEs with scalar-valued operators we extend this technique further to treat nonlinear coupled PDEs, again with the same scalar-valued operators. For this purpose, and only for this purpose, we introduce the so-called general form. The general form does not introduce any new operators, it merely reorganises them in a different way to be able to deal with coupled nonlinear equations in an easy manner. The general form is given by

$$\tau_2 \frac{\partial^2 u}{\partial t^2} + \tau_1 \frac{\partial u}{\partial t} + \nabla \Gamma = F, \tag{4.9}$$

with

$$\Gamma = -\sigma(u)\nabla u - \alpha(u)u + \gamma(u)$$

and

$$F = f(u) - \beta(u) \cdot \nabla u - a(u)u.$$

The linearisation of a, possibly coupled, nonlinear partial differential equation can be done by evaluating the following once [1]:

$$\begin{array}{lll}
\sigma_{ijkl} = -\frac{\partial \Gamma_{ik}}{\partial\left(\frac{\partial u_j}{\partial x_l}\right)} & \alpha_{ijk} = -\frac{\partial \Gamma_{ik}}{\partial u_j} & f = F \\
\beta_{ijk} = -\frac{\partial F_i}{\partial\left(\frac{\partial u_j}{\partial x_k}\right)} & a_{ij} = -\frac{\partial F_i}{\partial u_j} & \gamma = \Gamma,
\end{array}$$

with $\tilde{u} = u - u_0$ where u_0 is the linearisation point. With $i, j = 1, .., s$ and $k, l = 1, .., n$. s is the number of scalar equations (e.g. u and v) and n the number of independent variables (e.g. x and y) in $\mathbb{R}^n$. The resulting equations are linearised and can be looped over until convergence is reached.

Example: We consider the inviscid Burger's equation

$$\frac{\partial}{\partial t} u + u \nabla u = 0.$$

It can be linearised as follows. First we identify Γ and F

$$\begin{aligned} \Gamma &= 0 \\ F &= -\beta(u) \cdot \nabla u = u \cdot \nabla u \end{aligned}$$

We then set the space dimension to $n = 2$ which implies $\mathbb{R}^2$ and $k, l = 1, .., 2$. We have one equation implying $s = 1$. Thus, $i, j = 1$. Building the derivatives first for σ

$$\sigma_{ijkl} = -\frac{\partial \Gamma_{ik}}{\partial \left(\frac{\partial u_j}{\partial x_l} \right)} \overset{i,j=1}{\widehat{=}} \sigma_{11kl} = -\frac{\partial \Gamma_{1k}}{\partial \left(\frac{\partial u_1}{\partial x_l} \right)} = \begin{pmatrix} \frac{\partial \Gamma_{11}}{\partial \left(\frac{\partial u_1}{\partial x_1} \right)} & \frac{\partial \Gamma_{11}}{\partial \left(\frac{\partial u_1}{\partial x_2} \right)} \\ \frac{\partial \Gamma_{12}}{\partial \left(\frac{\partial u_1}{\partial x_1} \right)} & \frac{\partial \Gamma_{12}}{\partial \left(\frac{\partial u_1}{\partial x_2} \right)} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

Then for α we write

$$\alpha_{11k} = -\frac{\partial \Gamma_{1k}}{\partial u_1} = \begin{pmatrix} -\frac{\partial \Gamma_{11}}{\partial u_1} & -\frac{\partial \Gamma_{12}}{\partial u_1} \end{pmatrix} = \begin{pmatrix} 0 & 0 \end{pmatrix}.$$

Next for β we have

$$\beta_{11k} = -\frac{\partial F_1}{\partial \left(\frac{\partial u_1}{\partial x_k} \right)} = -\begin{pmatrix} \frac{\partial F_1}{\partial \left(\frac{\partial u_1}{\partial x_1} \right)} & \frac{\partial F_1}{\partial \left(\frac{\partial u_1}{\partial x_2} \right)} \end{pmatrix} = -\begin{pmatrix} \frac{\partial \left(u \left(\frac{\partial}{\partial x} u + \frac{\partial}{\partial y} u \right) \right)}{\partial \left(\frac{\partial u_1}{\partial x_1} \right)} & \frac{\partial \left(u \left(\frac{\partial}{\partial x} u + \frac{\partial}{\partial y} u \right) \right)}{\partial \left(\frac{\partial u_1}{\partial x_2} \right)} \end{pmatrix} = -\begin{pmatrix} u & u \end{pmatrix}.$$

And last for a

$$a_{11} = -\frac{\partial F_1}{\partial u_1} = \frac{\partial \left(u \left(\frac{\partial}{\partial x} u + \frac{\partial}{\partial y} u \right) \right)}{\partial u_1} = \left(\frac{\partial}{\partial x} u + \frac{\partial}{\partial y} u \right).$$

To sum up we note that the complexity and difficulty to find general solution procedures in both coupled and nonlinear partial differential equations can be handled with the simpler process of finding the correct operator's input functions.

References

[1] COMSOL. *COMSOL User's Guide 3.3.*

[2] COMSOL. *FEMLAB 3, User's Guide.*

[3] Herbert Ernst. *High-Resolution Thermal Measurements in Fluids.* PhD thesis, Universitaet Freiburg, 2001.

[4] Oliver Rübenkönig, Zhenyu Liu, and G. Jan Korvink. Integrated engineering development environment. *The Mathematica Journal*, 10, Issue 3:564 – 578, 2007.

5 Modelling Free Surface Flow with Operators

q. e. d.
quite easily done
(Unknown)

5.1 Flow: The Navier-Stokes Equation **44**
5.2 Free Surfaces: The Level-Set Method **47**
5.3 Coupling Components **51**

MODELING is the basis for design. In this section we show how an ink-jet model can be designed adopting the operator framework developed previously. To this end we first linearise the Navier-Stokes equation. In a next step we introduce implicit functions as a free surface representation. In combination with an advection equation this leads to the level-set method. The free surface representation will then be coupled to the Navier-Stokes equations. The coupling is dual; both the Navier-Stokes equations will receive information from the free surface via an external volume force and the advection of the free surface will be conducted through an advection equation powered by the velocity field obtained from the Navier-Stokes equation.

5.1 Flow: The Navier-Stokes Equation

5.1.1 Linearisation

The incompressible Navier-Stokes equation in vector form is given as

$$\begin{aligned} \rho \frac{\partial}{\partial t}\mathbf{u} + \rho \left(\mathbf{u}\cdot\nabla\right)\mathbf{u} - \nabla\mu\nabla\mathbf{u} + \nabla p &= \mathbf{f} \\ \nabla\cdot\mathbf{u} &= 0. \end{aligned} \tag{5.1}$$

μ is the dynamic viscosity, ρ is the density, $\mathbf{u}$ is the vector valued velocity field, p is the pressure and $\mathbf{f}$ is a vector valued volume force field. In component form the equation reads

$$\begin{aligned} \rho\frac{\partial}{\partial t}u + \rho\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) - \left(\frac{\partial}{\partial x}\mu\frac{\partial u}{\partial x} + \frac{\partial}{\partial y}\mu\frac{\partial u}{\partial y}\right) + \frac{\partial p}{\partial x} &= f_1 \\ \rho\frac{\partial}{\partial t}v + \rho\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) - \left(\frac{\partial}{\partial x}\mu\frac{\partial v}{\partial x} + \frac{\partial}{\partial y}\mu\frac{\partial v}{\partial y}\right) + \frac{\partial p}{\partial y} &= f_2 \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0. \end{aligned}$$

Since the Navier-Stokes equation is non-linear we rearrange the equation to fit into the general form. The general form in vector notation is given by

$$\tau_1\frac{\partial}{\partial t}\mathbf{u} + \nabla\cdot\mathbf{\Gamma} = \mathbf{F}.$$

The rearranged vector form of the Navier-Stokes equation reads

$$\begin{aligned} \rho\frac{\partial}{\partial t}\mathbf{u} + \nabla\cdot\left(-\mu\nabla\mathbf{u}\right) &= \mathbf{f} - \rho\left(\mathbf{u}\cdot\nabla\right)\mathbf{u} - \nabla p \\ 0 &= -\left(\nabla\cdot\mathbf{u}\right). \end{aligned}$$

The rearranged Navier-Stokes equation in component form reads as

$$\rho\frac{\partial}{\partial t}u + \left(\frac{\partial}{\partial x}\underbrace{\left(-\mu\frac{\partial u}{\partial x}\right)}_{\Gamma^{u_x}} + \frac{\partial}{\partial y}\underbrace{\left(-\mu\frac{\partial u}{\partial y}\right)}_{\Gamma^{u_y}}\right) = \underbrace{f_1 - \rho\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) - \frac{\partial p}{\partial x}}_{F^u}$$

$$\rho\frac{\partial}{\partial t}v + \left(\frac{\partial}{\partial x}\underbrace{\left(-\mu\frac{\partial v}{\partial x}\right)}_{\Gamma^{u_x}} + \frac{\partial}{\partial y}\underbrace{\left(-\mu\frac{\partial v}{\partial y}\right)}_{\Gamma^{u_y}}\right) = \underbrace{f_2 - \rho\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) - \frac{\partial p}{\partial y}}_{F^u}$$

$$0 = \underbrace{-\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right)}_{F^p}.$$

The identification of the terms in the general form delivers the following identities

$$\mathbf{\Gamma} = \begin{pmatrix} \Gamma^{u_x} & \Gamma^{u_y} \\ \Gamma^{v_x} & \Gamma^{v_y} \\ \Gamma^{p_x} & \Gamma^{p_y} \end{pmatrix} = \begin{pmatrix} -\mu\frac{\partial u}{\partial x} & -\mu\frac{\partial u}{\partial y} \\ -\mu\frac{\partial v}{\partial x} & -\mu\frac{\partial v}{\partial y} \\ 0 & 0 \end{pmatrix}$$

and

$$F = \begin{pmatrix} F^u \\ F^v \\ F^p \end{pmatrix} = \begin{pmatrix} f_1 - \rho\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) - \frac{\partial p}{\partial x} \\ f_2 - \rho\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) - \frac{\partial p}{\partial y} \\ -\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \end{pmatrix}.$$

Now we have identified the terms of the general form. To automatically find a linearisation we transform the general form to the coefficient form. The coefficient form is given by

$$\tau_2\frac{\partial^2}{\partial t^2}u + \tau_1\frac{\partial}{\partial t}u + \nabla^T\cdot(-\sigma\nabla u - \alpha u + \gamma) + \beta\cdot\nabla u + au = f.$$

The coefficient form corresponds to the general form with

$$\Gamma = -\sigma\nabla u - \alpha u + \gamma$$

and

$$F = f - \beta\cdot\nabla u - au.$$

The transformation form general to coefficient form is done according to the later rules:

$$\begin{array}{lll} \sigma_{ijkl} = -\frac{\partial\Gamma_{ik}}{\partial\left(\frac{\partial u_j}{\partial x_l}\right)} & \alpha_{ijk} = -\frac{\partial\Gamma_{ik}}{\partial u_j} & \gamma_{ij} = \Gamma_{ij} \\ \beta_{ijk} = -\frac{\partial F_i}{\partial\left(\frac{\partial u_j}{\partial x_k}\right)} & a_{ij} = -\frac{\partial F_i}{\partial u_j} & f_i = F_i. \end{array}$$

Which deliver the following input functions:

$$\sigma = \begin{pmatrix} \begin{pmatrix} \mu & 0 \\ 0 & \mu \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} \mu & 0 \\ 0 & \mu \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{pmatrix}$$

$$\alpha = \mathbf{0}$$

$$\gamma = \begin{pmatrix} -\mu\frac{\partial u}{\partial x} & -\mu\frac{\partial u}{\partial y} \\ -\mu\frac{\partial v}{\partial x} & -\mu\frac{\partial v}{\partial x} \\ 0 & 0 \end{pmatrix}$$

$$\beta = \begin{pmatrix} \begin{pmatrix} \rho u & \rho v \end{pmatrix} & \begin{pmatrix} 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \end{pmatrix} & \begin{pmatrix} \rho u & \rho v \end{pmatrix} & \begin{pmatrix} 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \end{pmatrix} \end{pmatrix}$$

$$\mathbf{a} = \begin{pmatrix} \rho\frac{\partial u}{\partial x} & \rho\frac{\partial u}{\partial y} & 0 \\ \rho\frac{\partial v}{\partial x} & \rho\frac{\partial v}{\partial y} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{f} = \mathbf{F}.$$

Re-inserting the above into the coefficient form results in the linearised Navier-Stokes equations. The linearised equations in component form read as follows:

$$\begin{aligned}
\rho\frac{\partial}{\partial t}u - \underbrace{\left(\frac{\partial}{\partial x}\mu\frac{\partial u}{\partial x} + \frac{\partial}{\partial y}\mu\frac{\partial u}{\partial y}\right)}_{\sigma_{11}} + \rho\underbrace{\left(u^k\frac{\partial u}{\partial x} + v^k\frac{\partial u}{\partial y}\right)}_{\beta_{11}} + \underbrace{\frac{\partial p}{\partial x}}_{\beta_{13}} + \underbrace{\rho\frac{\partial u}{\partial x}^k}_{a_{11}} u + \underbrace{\rho\frac{\partial u}{\partial y}^k}_{a_{12}} v &= \\
\underbrace{f_1 - \rho\left(u^k\frac{\partial u}{\partial x}^k + v^k\frac{\partial u}{\partial y}^k\right) - \frac{\partial p}{\partial x}^k}_{F_1} + \underbrace{\frac{\partial}{\partial x}\mu\frac{\partial u}{\partial x}^k}_{\gamma_{11}} + \underbrace{\frac{\partial}{\partial y}\mu\frac{\partial u}{\partial y}^k}_{\gamma_{12}} & \\
\rho\frac{\partial}{\partial t}v - \underbrace{\left(\frac{\partial}{\partial x}\mu\frac{\partial v}{\partial x} + \frac{\partial}{\partial y}\mu\frac{\partial v}{\partial y}\right)}_{\sigma_{22}} + \rho\underbrace{\left(u^k\frac{\partial v}{\partial x} + v^k\frac{\partial v}{\partial y}\right)}_{\beta_{22}} + \underbrace{\frac{\partial p}{\partial y}}_{\beta_{23}} + \underbrace{\rho\frac{\partial v}{\partial x}^k}_{a_{21}} u + \underbrace{\rho\frac{\partial v}{\partial y}^k}_{a_{22}} v &= \\
\underbrace{f_2 - \rho\left(u^k\frac{\partial v}{\partial x}^k + v^k\frac{\partial v}{\partial y}^k\right) - \frac{\partial p}{\partial y}^k}_{F_2} + \underbrace{\frac{\partial}{\partial x}\mu\frac{\partial v}{\partial x}^k}_{\gamma_{21}} + \underbrace{\frac{\partial}{\partial y}\mu\frac{\partial v}{\partial y}^k}_{\gamma_{22}} & \\
\underbrace{\frac{\partial u}{\partial x}}_{\beta_{31}} + \underbrace{\frac{\partial v}{\partial y}}_{\beta_{32}} &= \\
\underbrace{-\left(\frac{\partial u}{\partial x}^k + \frac{\partial v}{\partial y}^k\right)}_{F_3} &
\end{aligned}$$

Note that the γ parts are on the right hand side. γ is independent of the unknown u and may be moved to the right hand side of the equation.

5.1.2 Boundary conditions and constraints

After the discretisation we are left with the tangential stiffness matrix

$$K(u) = -\frac{\partial L(u)}{\partial U} = \begin{bmatrix} K_{uu} & K_{uv} & K_{up} \\ K_{vu} & K_{vv} & K_{pv} \\ K_{pu} & K_{pv} & K_{pp} \end{bmatrix}$$

where

$$\frac{\partial L_u}{\partial u} = K_{uu} \text{ et cetera.}$$

In our notation K corresponds to the left hand side of the equation and L to the right hand side. To account for the boundary conditions we use Lagrangian multipliers and extend the system of equations in the following manner

$$\underbrace{\begin{bmatrix} K(u^k) & N^T \\ N & 0 \end{bmatrix}}_{K_e} \cdot \begin{bmatrix} \delta u \\ \Lambda \end{bmatrix} = \underbrace{\begin{bmatrix} L(u^k) - N^T \cdot \Lambda^k \\ M - N \cdot u^k \end{bmatrix}}_{L_e}.$$

Here Λ are the Lagrangian multipliers. N is the Jacobian of M. That is $N = -{}^{\partial M}/_{\partial U}$. M are the boundary values.

The update is then done by

$$u^{k+1} = u^k + \lambda \delta u.$$

Here λ is a damping factor. λ must be $0 < \lambda \leq 1$ and is computed with the affine invariant newton method [4, 6] as outlined on page 113.

5.2 Free Surfaces: The Level-Set Method

In the context of computational fluid dynamics free surfaces are defined to be the interface between two or more materials of different physical property.

The interface region is assumed to be infinitely small, there is no transition region as such which is a simplification. The domain Ω under inspection can thus be divided in three distinct parts

$$\cup\,\Omega = \Omega_{\text{Outside}} + \Gamma_{\text{Interface}} + \Omega_{\text{Inside}}$$

To represent $\Gamma_{\text{Interface}}$ (Γ_{I}) explicit and implicit approaches exist.

5.2.1 Implicit functions

To represent Γ_I two methods are in common use. Explicit and implicit representations. In an explicit approach we work with coordinates explicitly describing the interface location. In the implicit approach we utilise an implicit function to locate the interface. In the explicit representation a circle in $\mathbb{R}^2$ can be portrayed through a collection of coordinates in 2-dimensional space. Between these coordinates an interpolation is then assumed and therewith the points connected which represents Γ_I. In the context of free surface flow this technique leads to the volume of fluid method (VOF) [5]. Here the cell fractions c ranging from 0 to 1 are stored in a mesh which is put over the computational domain. From these cell fractions the explicit surface is reconstructed. This presents the technique used by most commercial free surface simulation tools.

In the implicit approach we use a scalar valued function such as

$$\phi(x, y) = \sqrt{(x - x_o)^2 - (y - y_0)^2} - R \tag{5.2}$$

to represent the interface. Where $\phi : \mathbb{R}^2 \mapsto \mathbb{R}$ and x_0, y_0 is the centre of a circle and R the circle's radius. Each computed value for x and y can be understood as a height value. We group these height values together to ϕ_c to make a set of contours. Γ_I is typically represented by ϕ_0, the 0-contour. We define

$$\phi(x, y) = \begin{cases} 0 & \Gamma_\mathrm{I} \\ < 0 & \Omega_\mathrm{Inside} \\ > 0 & \Omega_\mathrm{Outside} \end{cases}$$

See Figure 5.1 for the representation of a circle by an explicit and an implicit method. To be useful the surface representation must be easily submittable to fundamental computational geometry operations such as finding the union, intersection and complement of several representations. Such operations are based on intersecting convex polygons. In [7] an algorithm for intersecting convex polygons in $\mathbb{R}^2$ can be found. The algorithm can be extended to operations such as finding the convex union and complement. Each concave polygon can be split to several convex polygons; this endeavour is everything else but trivial in $\mathbb{R}^3$.

Not so for implicit representations. Modelling with implicit functions as done in Bloomenthal [1, 10] can be accomplished by using min, max and -1. For two spheres

$$\begin{aligned} s_1 &= \sqrt{\left(x - \frac{1}{2}\right)^2 + y^2 + z^2} - 1 \\ s_2 &= \sqrt{\left(x + \frac{1}{2}\right)^2 + y^2 + z^2} - 1 \end{aligned}$$

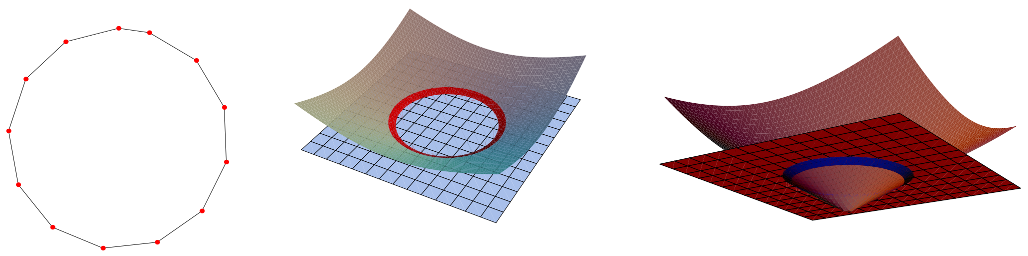


Figure 5.1: Explicit and implicit representation of a circle in $\mathbb{R}^2$. The explicit representation (left) is based on connecting coordinates to lines to form the interface Γ_I. In the implicit approach a function such as $\phi(x, y) = \sqrt{x^2 - y^2} - \frac{1}{2}$ cuts the $\mathbb{R}^2$ plane, which represents Γ_I. Middle: Viewing from $\mathbb{R}^+$ showing $\phi > 0$ with a red band marking values $0 \le \phi \le 0.05$. Left: Viewing from $\mathbb{R}^-$ showing $\phi < 0$ with a blue band marking $-0.05 \le \phi \le 0$.

the intersection of $s_1 \cap s_2$

$$min(s_1, s_2)$$

the union of $s_1 \cup s_2$ is

$$max(s_1, s_2).$$

and the free cut of s_2 from s_1 is

$$min(s_1, -s_2)$$

This is depicted in Figure 5.2.

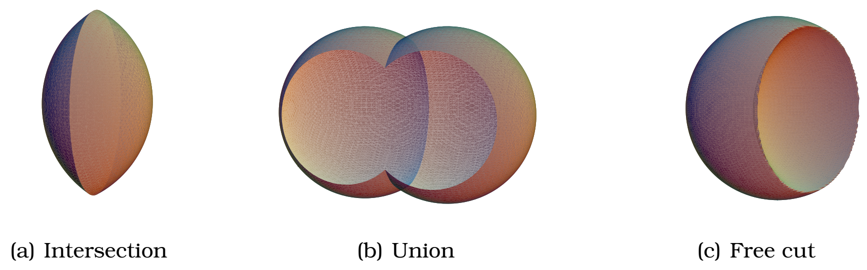


Figure 5.2: Basic modelling operations for implicit functions. Left we have an intersection of two spheres, in the middle a union and to the right a free cut.

5.2.2 Dynamics of implicit functions - level set

Free surfaces, no matter which representation is used, are advected with

$$\frac{\partial}{\partial t}\phi + \mathbf{v} \cdot \nabla\phi = 0 \tag{5.3}$$

The combination of implicit functions such as 5.2 with 5.3 is called Level Set Method [8, 9]. Here $\mathbf{v}$ is the driving velocity field, later obtained from the solution of the Navier-Stokes equation.

5.2.3 Surface tension

Brackbill showed that the surface tension can be modelled as volume force [2]

$$\mathbf{f}_{st} = \gamma\kappa\delta\mathbf{n}$$

with γ the constant surface tension coefficient, κ the curvature, δ is the Dirac distribution and $\mathbf{n}$ is the normal to the interface. κ, $\mathbf{n}$ and δ are functions of ϕ. We will investigate them separately. The computation of the normal $\mathbf{n}$ can be done via

$$\mathbf{n} = \left.\frac{\nabla\phi}{|\nabla\phi|}\right|_{\phi=0}$$

$$\begin{aligned} n_x &= \frac{\phi_x}{\sqrt{\phi_x^2 + \phi_y^2}} \\ n_y &= \frac{\phi_y}{\sqrt{\phi_x^2 + \phi_y^2}}. \end{aligned}$$

The computation of the delta distribution we proceed along the lines of Tornberg [11, 12]. We replace δ with δ_{h_w} where δ_{h_w} is a smooth function with support width h_w and a measure for δ. One possible choice is the derivative of a smoothed Heaviside function (see Chapter 6.3.6).

To compute the curvature we might use [2]

$$\kappa = \nabla^T \cdot \left.\left(\frac{\nabla\phi}{|\nabla\phi|}\right)\right|_{\phi=0}.$$

If we can guarantee the signed distance property of ϕ in Ω then

$$|\nabla\phi| = 1|_{\Omega}$$

and thus

$$\kappa = \nabla^T \cdot (\nabla\phi)\big|_{\phi=0}.$$

This formulation has the distinct disadvantage that we need to compute the second derivative of ϕ to compute κ. The surface tension can also be written as

[3]

$$
\begin{aligned}
\gamma\kappa\delta\mathbf{n} &= \nabla\cdot\mathbf{T} \\
\mathbf{T} &= \gamma\left(\mathbf{I}-\left(\mathbf{n}\mathbf{n}^T\right)\right)\delta
\end{aligned}
$$

where $\mathbf{I}$ is the identity matrix. In this formulation we avoid computing the second derivative, previously necessary for the computation of κ.

5.3 Coupling Components

Coupling the Navier-Stokes equation and the Level Set equation results in

$$
\begin{aligned}
\rho\frac{\partial}{\partial t}\mathbf{u}-\nabla^T\mu\nabla\mathbf{u}+\rho(\mathbf{u}\cdot\nabla\mathbf{u})+\nabla p+\gamma\kappa\delta\mathbf{n} &= \mathbf{f} \\
\nabla\mathbf{u} &= 0 \\
\frac{\partial}{\partial t}\phi+\mathbf{u}\cdot\nabla\phi &= 0.
\end{aligned}
$$

In this work we proceed as follows: First, we compute the Navier-Stokes equation with the load $\mathbf{f}$ which includes the surface tension force $\mathbf{f}_{st}$. κ, δ, $\mathbf{n}$ and $\mathbf{f}_{st}$ are dependent on ϕ. The resulting velocity field $\mathbf{u}$ is then, in a second step, used to time integrate the level set equation and thus compute the new location of ϕ. In a third step the implicit function ϕ is re-initialised and/or re-set.

References

[1] J. Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan-Kaufmann, 1987.

[2] Brackbill, Kothe, and Zemach. A continuum method for modeling surface tension. *J. Comput. Phys*, 100:335 – 354, 1992.

[3] COMSOL. *COMSOL User's Guide 3.3*.

[4] Peter Deuflhard. A modified newton method for the solution of ill-conditioned systems of nonlinear equations with appication to multiple shooting. *Numerische Mathematik*, 22:289 – 315, 1974.

[5] C. W. Hirt and B. D. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of Computational Physics*, 39:201 – 225, 1981.

[6] Nowak and Weimann. A family of newton codes for systems of highly nonlinear equations. *Konrad-Zuse-Zentrum*, Technical Report TR-91-10:na, 1991.

[7] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2002.

[8] Stanley J. Osher and Ronald P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.

[9] J.A. Sethian. *Levelset methods and fast marching methods*. Cambridge University Press, 1999.

[10] Duane Storti, Cornelius Nevrinceanum, and Mark Ganter. A tutorial on implicit solid modeling. *The Mathematica Journal*, 2(Issue 3):70 – 78, 1992.

[11] A.-K. Tornberg and B. Engquist. Interface tracking in multiphase flows, 2002.

[12] Anna-Karin Tornberg and Björn Engquist. Regularization techniques for numerical approximation of pdes with singularities. *Journal of Scientific Computing*, 19, Issue 1–3:527 – 552, 2003.

6 Operator Discretisation

A mathematician is a machine that turns coffee into theorems.
(Paul Erdös)

6.1 Lumped Operators **54**
6.2 Weighted Residual Method **58**
6.3 Finite Element Method **62**
6.4 Initial & Boundary Conditions **85**

DISCRETISATION is the process where continuous mathematical formulae are adapted for computer processing. To this end the continuous formulae are transformed into discrete formulae which are suitable for computer implementation. An algorithm[1] is then found since this is in effect a re-written discretised formula.

[1]The word algorism comes from the name al-Khwarizmi - "the one from Khwarizm" - of an early 9th century Persian mathematician, possibly from what is now Khiva in western Uzbekistan. The word evolved into the modified form algorithm, with a generalisation of the meaning to any set of rules specifying a computational procedure. Occasionally algorism is also used in this generalised meaning, especially in older texts.

The discussion is divided into four sections. First the lumped operators are discussed briefly. In the second section we introduce the weighted residual method as a general framework for discretising partial differential equations. We discuss several possible discretisation methods and support the theory with an example. The third section, the largest part of the discussion, sheds some light on the finite element method. In the fourth and last section we present both boundary and initial conditions, both, in a general light and some specifics encountered in fluid settings.

6.1 Lumped Operators

In this section we present stamps for the lumped circuit operators. We start with the basic elements and move to more advanced elements later. The stamps are presented in the following manner

Matrix Type	i j $\dots$
i	
j	values,
$\vdots$	

where the "Matrix Type" represents the system matrices the stamps are built into. The options are the stiffness matrix S, the damping matrix D, the mass matrix M, and the load vector L. Depending on the type of analysis to be performed on the lumped system of equations the lumped system has to be assembled in a different manner. Due to this three destination matrices are given as Matrix Type, first the transient case, then the harmonic case, and last the stationary case. $D/S/S$ indicates that the considered stamp is built into the damping matrix if a transient analysis is to be performed and into the stiffness matrix if a harmonic or stationary analysis is required. Usually, in a system suitable for a transient analysis all the matrices are populated. Stationary and harmonic analyses get by with the stiffness matrix and load vector. The indices $i, j, ...$ specify where in the global matrices the stamps are to be inserted. See Chapter 2 for more details on the matrix assembly process. Some of the stamps require that the system of equations be enlarged. This is indicated by using Greek indices.

6.1.1 Basic R, L, and C elements

We start with the R, L, and C stamps. Each element is connected to nodes i and j. The basic R, L and C stamps

$D/S/S$	i	j
i	$\frac{1}{R}$	$-\frac{1}{R}$
j	$-\frac{1}{R}$	$\frac{1}{R}$

$S/S/-$	i	j
i	$\frac{1}{L}$	$-\frac{1}{L}$
j	$-\frac{1}{L}$	$\frac{1}{L}$

$M/S/-$	i	j
i	C	$-C$
j	$-C$	C

then give the entries for the global element matrix. For a transient analysis the resistive parts belong to the damping matrix, the inductive parts belong to the stiffness matrix, and the capacitive parts to the mass matrix.

For a harmonic analysis the capacitive elements are multiplied by a harmonic factor $\imath\omega$ and the inductive parts by $1/(\imath\omega)$. Here $\imath$ is the imaginary unit. The resistive parts are not subject to change. Each component is then built into the stiffness matrix. Other methods for setting up harmonic systems are as well possible.

For a stationary analysis the capacitances are removed from the system altogether and inductances are treated as wires, since the capacitances are a break of the circuit and the inductances are a short circuit. With this in mind we can then directly form the system of equations where the resistive parts are inserted into the stiffness matrix.

6.1.2 Ground and wires

Ground has no stamp but imposes a Dirichlet boundary condition with value 0, however, all other values are possible. A wire can be treated as a special case of a voltage source, were the voltage U is 0.

6.1.3 Uncontrolled sources

Generally speaking, we have two different kinds of sources. Controlled and uncontrolled sources. We begin with the uncontrolled sources, namely current and voltage source.

Current source

The current can be discretised as in

L	i
i	I
j	$-I$

Here we note that the current source is built into the load vector.

Voltage source

By contrast, the voltage source is made up of two parts. One part goes into the load vector, the other part goes into the damping matrix for a transient analysis and into the stiffness matrix for other kinds of analyses. The voltage source can be discretised as

$D/S/S$	i	j	κ
i	0	0	1
j	0	0	-1
κ	1	-1	0

L	i
i	0
j	0
κ	U

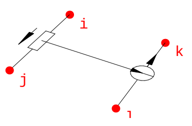
The voltage source enlarges the system of equations and adds a new row and column at position κ and the actual voltage U is inserted into the load vector at position κ.

6.1.4 Controlled sources

We have four different kinds of controlled sources. All of them work by introducing a ratio of the output source to the input source and thus no modification of the load is necessary. Controlled sources can, for example, be used to model transistors.

Voltage controlled current source

First, we present the voltage controlled current source. In this device the output current between nodes k and l depends on the voltage applied at the input terminals at nodes i and j. The two are related by the transconductance g which is the ratio of the output current to the input voltage and is measured in Siemens[2].

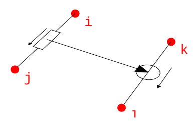


$D/S/S$	i	j	k	l
i	0	0	0	0
j	0	0	0	0
k	g	$-g$	0	0
l	$-g$	g	0	0

Voltage controlled voltage source

Second, we have the voltage controlled voltage source where the output voltage at nodes k and l depends on the input voltage at nodes i and j. The ratio of the output voltage to the input voltage determines the voltage gain A. This device enlarges the system of equations and the solution vector. Position μ of the solution vector gives the current through the output nodes k and l.

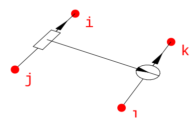
[2] Ernst Werner von Siemens (1816-1892) invented, among other things, a telegraph that used a needle to point to the right letter, instead of using Morse code.



$D/S/S$	i	j	k	l	μ
i	0	0	0	0	0
j	0	0	0	0	0
k	0	0	0	0	1
l	0	0	0	0	-1
μ	$-A$	A	1	-1	0

Current controlled current source

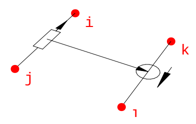
Next, the current controlled current source is presented. The output current at the nodes k and l of this source depends on the current through the input nodes i and j. The relating ratio of the input current to the output current is the current gain F.



$D/S/S$	i	j	k	l	μ
i	0	0	0	0	1
j	0	0	0	0	-1
k	0	0	0	0	F
l	0	0	0	0	$-F$
μ	1	-1	0	0	0

Current controlled voltage source

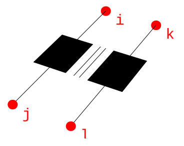
And last, we introduce the current controlled voltage source. The input current at nodes i and j is related to the output voltage at nodes k and l by R. This parameter is called the transresistance, which is the ratio of the output voltage to the input current and is measured in Ohm.



$D/S/S$	i	j	k	l	μ	ν
i	0	0	0	0	0	1
j	0	0	0	0	0	1
k	0	0	0	0	1	0
l	0	0	0	0	-1	0
μ	0	0	1	-1	0	$-R$
ν	1	-1	0	0	0	0

6.1.5 Mutual inductance

We present the discretised version of a mutual inductance. A mutual inductance couples two coils with inductances L_1 and L_2 by a coupling coefficient K. L_1 connects the nodes i and j and L_2 connects k and l. For $K = 1$ we have 100% coupling and all energy is transferred. $K = 0$ implies no coupling. K is the ratio of the mutual inductance M to the square root of the inductances. We thus have $0 < K = \frac{M}{\sqrt{L_1 L_2}} < 1$.

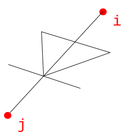


$S/S/-$	i	j	k	l
i	$\frac{1}{(1-K^2)L_1}$	$-\frac{1}{(1-K^2)L_1}$	$\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$	$-\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$
j	$-\frac{1}{(1-K^2)L_1}$	$\frac{1}{(1-K^2)L_1}$	$-\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$	$\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$
k	$\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$	$-\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$	$\frac{1}{(1-K^2)L_2}$	$-\frac{1}{(1-K^2)L_2}$
l	$-\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$	$\frac{1}{(-1+K^2)\sqrt{L_1 L_2}}$	$-\frac{1}{(1-K^2)L_2}$	$\frac{1}{(1-K^2)L_2}$.

The destination matrices are the same as for an ordinary inductance and so is the multiplication with the harmonic factor $\imath\,\omega$.

6.1.6 Non-linear elements

Diodes are non-linear, but can nevertheless be discretised by



L	i
i	$-\left(-1-\exp^{\frac{U_{d_i}-U_{d_j}}{n\cdot U_{th}}}\right) I_s$
j	$\left(-1-\exp^{\frac{U_{d_i}-U_{d_j}}{n\cdot U_{th}}}\right) I_s$.

Here U_d is the voltage across the diode. This voltage dependency is what makes the diode non-linear. I_s is the saturation current and U_{th} is the thermal voltage. The emission coefficient is n.

In a similar manner transistors or even operation amplifiers can be modelled in their non-linear region. An alternative is to use the basic diode element provided here, so that a library can be built on top of it. Non-linear elements, however, require a good non-linear system solver.

6.2 Weighted Residual Method

The weighted residual method represents a unification of many approximate solution methods for partial differential equations. The following outline parallels the treatments of [1, 4].

With the weighted residual method we try to find approximate continuous solutions to partial differential equations. We have

$$\begin{aligned} \mathcal{L}(u) &= 0 \\ \mathcal{B}(u) &= 0. \end{aligned}$$

where $\mathcal{L}(\cdot)$ is a general differential operator in the domain Ω and $\mathcal{B}(\cdot)$ on the boundary $\partial\Omega$. We assume a trial solution of the form

$$u = u_{\partial\Omega}(\mathbf{x}, t) + \sum_{j=1}^{N} u_j(t)\alpha_j(\mathbf{x}) \tag{6.1}$$

where u is an approximation to u_e, the exact solution. The $\alpha_j(\mathbf{x})$ are chosen, analytical functions, so-called "Ansatzfunktionen". The $u_j(t)$ represent unknown constants which we would like to compute. $u_{\partial\Omega}(\mathbf{x}, t)$ is a known function that satisfies the essential boundary conditions. In general this means that

$$u_{\partial\Omega} = g,\ \alpha_j = 0,\ \forall \mathbf{x} \text{ on } \partial\Omega.$$

The finite summation implies the transition from the continuous to a discrete representation.
The residual[3] is defined as

$$r_{\Omega} = \mathcal{L}(u) - f \tag{6.2}$$

and the boundary residual $r_{\partial\Omega}$ as

$$r_{\partial\Omega} = \mathcal{B}(u) - g.$$

The residual is a measure of the extent to which function u satisfies the differential equation. For $N \to \infty$ the hope is that $r_{\Omega}, r_{\partial\Omega} \to 0$. The exact solution is obtained when both residuals are zero. This would be the ideal case. This also means that each of the N components of r_{Ω} and $r_{\partial\Omega}$ are equal to zero. We can soften this requirement in the sense that we require that r_{Ω} and $r_{\partial\Omega}$ are zero in an average sense. To do so we set the inner product of r_{Ω} and some prescribed weighting function w_i for $i = 1, .., N$ to zero

$$\langle r_{\Omega},\, w_i\rangle = 0 \text{ and } \langle r_{\partial\Omega},\, \hat{w}_i\rangle = 0,$$

where

$$\langle u,\, v\rangle = \int_{\Omega} uv \, d\Omega.$$

In the case where $u = u_e$ the equation is still satisfied. Thus we set the integral

[3]The residual is not to be confused with the error which is defined as $\|u - u_e\|_p$ in some norm p. Note that the residual is readily computable; the error is in most cases not amenable.

of the weighted residuals equal to zero

$$\int_{\Omega} r_{\Omega} w_i \;\, d\Omega + \int_{\partial\Omega} r_{\partial\Omega} \hat{w}_i \;\, d\partial\Omega = 0. \tag{6.3}$$

Requiring that the integrals are satisfied $\forall i$ as $N \to \infty$ can be viewed as requiring that $u \to u_e$ for $N \to \infty$. So, if the weighted residual $\int_{\Omega} r_{\Omega} w_i \, d\Omega = 0 \; \forall i$, then r_{Ω} is minimal with respect to the chosen Ansatzfunktionen [1].

Now we substitute Equation 6.1 in Equation 6.2

$$r_{\Omega} = \mathcal{L}\left(u\right) - f = \mathcal{L}\left(u_{\partial\Omega}\left(\mathbf{x}, t\right) + \sum_{j=1}^{N} u_j(t)\alpha_j\left(\mathbf{x}\right) \right) - f, \tag{6.4}$$

where we have $j = 1, .., N$ unknowns. Since $u_j(t)$ are constants we can write in matrix form

$$A\mathbf{u} = b,$$

where we have inserted Equation 6.4 in Equation 6.3

$$\begin{aligned} A_{ij} &= \int_{\Omega} \mathcal{L}\left(u_j\right) w_i \, d\Omega \\ b_i &= \int_{\Omega} f w_i \, d\Omega + \int_{\partial\Omega} \mathcal{L}\left(u_{\Gamma}(\mathbf{x}, t)\right) \hat{w}_i \, d\partial\Omega. \end{aligned}$$

Further detail can be seen in the example below.

Usually $w_i = \hat{w}_i$. The essential question is the choice of the weighting function w_i. The weighting function can be chosen in several different ways where $i = 1, .., N$:

1. Collocation method (leading to the Finite Difference Method)

   In this approach we select as many points as there are undetermined parameters i and choose the parameters to ensure that the residual is zero at these points - which does not imply that the approximate solution will be equal to the exact solution at these points - the error may still be substantial. The weighting functions are Dirac delta distributions,

   $$w_i = \delta(x - x_i).$$

2. Sub-domain method (leading to the Finite Volume Method)

   In this method we subdivide the simulation domain Ω into as many sub-intervals Ω_i as there are free parameters so that the average value of the residual over each sub-interval is zero

   $$w_i = \left\{ \begin{array}{ll} 1; & \forall i \in \Omega_i \\ 0; & \text{else} \end{array} \right. .$$

3. Interior method (leading to the Boundary Element Method)

   Here we assume the weighting function is similar to the solution. The weights are so called "Green's functions", and the nodes (where the centre of these functions is located) are placed on the material boundaries [2].

4. Least squares method

   Here the weighting is chosen

$$w_i = \frac{\partial r_\Omega}{\partial \alpha_i},$$

   where the residual is differentiated with respect to the Ansatzfunktionen. This can be understood as the minimisation of an energy norm. The resulting system of equations is, however, often ill conditioned [4].

5. Bubnov-Galerkin[4] method (leading to the Finite Element Method)

   In this approach the weighting functions are chosen to be the same as the Ansatzfunktionen

$$w_i = \alpha_i$$

   We look at this method in further detail in the example further down.

6. Petrov-Galerkin

   Here the weighting function is not chosen to be the Ansatzfunktion but

$$w_i = h_i,$$

   where h_i is an analytic function with additional terms to further constrain the approximate solution. This is frequently used if the convective part of a partial differential equation is dominant compared to the diffusive part. The Petrov-Galerkin method is a generalisation of the Bubnov-Galerkin method.

The following example is to illustrate the procedure so far derived.

Example: We investigate the weighted residual method with a Bubnov-Galerkin approach. We consider 1-dimensional heat conduction [4]. In the domain $\Omega \in [0, L]$ we have the partial differential equation

$$\frac{\partial^2}{\partial x^2} u(x) + f(x) = 0$$

and on $\partial\Omega$ the boundary conditions

$$u(0) = u(1) = 0.$$

[4]In recent papers and books the traditional name “Galerkin method” is replaced by the name Bubnov-Galerkin method to account for both independent contributors to the method.

Here we have

$$f(x) = x.$$

The equation has the exact solution

$$u_e = \frac{1}{6}\left(L^2 - x^2\right)x.$$

We choose $\alpha_j = \sin(j\pi x/L)$ and $w_i = \alpha_i$ with $\mathcal{L}_1(u) = \frac{\partial^2}{\partial x^2}u$ and $\mathcal{L}_2(u) = x$. For simplicity and without restriction of generality we set $L = 1$. Then we have for $N = 2$

$$A_{ij} = \int_\Omega \mathcal{L}_1(\alpha_j) w_i \, d\Omega = \begin{pmatrix} -\frac{\pi^2}{2} & 0 \\ 0 & -2\pi^2 \end{pmatrix}$$

and

$$b_i = \int_\Omega \mathcal{L}_2(\alpha_j) w_i \, d\Omega = \begin{pmatrix} -\frac{1}{\pi} \\ \frac{1}{2\pi} \end{pmatrix}.$$

The solution is thus

$$u_j = \begin{pmatrix} \frac{2}{\pi^3} & -\frac{1}{4\pi^3} \end{pmatrix}$$

and the sum according to Equation 6.1 is then

$$u = 0 + \sum_{j=1}^{2} u_j \alpha_j = \frac{2\sin(\pi x)}{\pi^3} - \frac{\sin(2\pi x)}{4\pi^3}.$$

The l_2 error norm can be computed as

$$||u - u_e||_2 \approx 0.0019.$$

Remark: The fact that A is diagonal is a coincidence. Choosing a polynomial for α_j, for example a series expansion with

$$\alpha_j = \sin(j\pi x/L) \approx \frac{j\pi x}{L} - \frac{\left(j^3\pi^3\right)x^3}{6L^3} + \frac{j^5\pi^5 x^5}{120L^5} - \frac{\left(j^7\pi^7\right)x^7}{5040L^7} + \frac{j^9\pi^9 x^9}{362880L^9} + O\left(x^{10}\right)$$

will also deliver a result. Now A is, however, fully populated, which is not wanted since computational complexity in time and space will rise during the inversion of A.

6.3 Finite Element Method

In this section we introduce shape functions which will help to overcome some of the shortcomings of the example presented in the weighted residual section. Next we derive the discretised equations for the partial differential equation operators. This is then followed by a discussion of numerical integration. Some aspects of meshes are presented. Furthermore, numerical challenges such as recovery of derivatives, regularisation, and re-initialisation are discussed.

6.3.1 Shape functions

Independent of the discussion of the weighted residual method, we consider the problem of finding an interpolation to a set of data points [2]. This can be achieved by Equation 6.1 and using a least squares method [8] for the coefficients. By increasing the degree of the polynomial the unknown data points are fitted with a higher accuracy. Unfortunately, the polynomial may oscillate unacceptably between the data points. Polynomials, however, have the desirable property that they are easily differentiable. As a remedy we give up the global interpolation intention and we subdivide the data points into subsections. To do so the domain Ω is split into non-overlapping elements where

$$\Omega \approx \cup_{e=1}^{n_\Omega} \Omega_e \text{ and } \partial\Omega \approx \cup_{i=1}^{n_{\partial\Omega}} \partial\Omega_e,$$

with n_Ω and $n_{\partial\Omega}$ the number of elements and the number of boundary segments respectively. The splitting of the domain Ω may not always be possible exactly. The elements are connected at their nodes to each other and form a mesh.

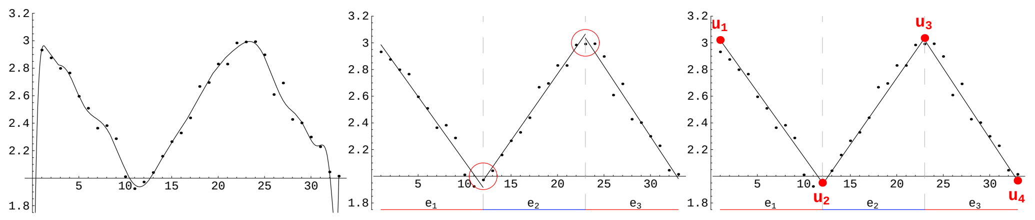


Figure 6.1: Left: Data interpolation with a high order polynomial on a global level. Middle: A subdivision into three elements and a interpolation with low order polynomials. The interpolation is discontinuous as shown by the red circles. Right: C^0 continuous local interpolation based on three elements and four additional nodes where values $u_1, .., u_4$ have been introduced.

Now we interpolate in those elements with low order polynomials as shown in Figure 6.1. This local interpolation may not be continuous at the element boundaries and to enforce at least C^0 continuity[5] additional constraints are necessary. To do so we define a linear variation between two values u_i and u_j

$$u(r) = N_i(r)u_i + N_j(r)u_j$$

with

[5]We speak about C^0 continuity if two functions are continuously joined. We speak of C^1 continuity if also the derivatives of first order are continuous across the interface.

$$N_i(r) = \frac{(1-r)}{2} \text{ and } N_j(r) = \frac{(1+r)}{2} \quad \forall r \in [-1,1].$$

Here r is a scaled, local coordinate of the form

$$r = \frac{x - (x_i + x_j)/2}{(x_j - x_i)/2}$$

with x_i and x_j the left and right global coordinate of element e_k respectively. We call $N(\mathbf{r}_i)$ shape functions where $\mathbf{r}_i$ is a scaled coordinate in a so-called mother element. Line elements use the domain $\Omega^{(e_k)} \in [-1,1]$ as mother element.

For a function to be a shape function the following must hold in the element's domain $\Omega^{(e_k)}$:

$$\sum_{i=1}^{n} N_i = 1,$$

which is known [13] as the partition of unity and the second condition

$$N_i(\mathbf{r}_j) = \delta_{ij} = \begin{cases} 1; & i = j \\ 0; & i \neq j \end{cases},$$

which states that the shape function N_i has a value of unity at node $\mathbf{r}_i$ and is zero at all other nodes. This is shown in the triangle mesh depicted in Figure 6.2. At the blue node for each red element the linear shape functions are unity and zero everywhere else. The weighting functions from the dashed black elements do not contribute to the global matrix, only weighting functions in red elements contribute.

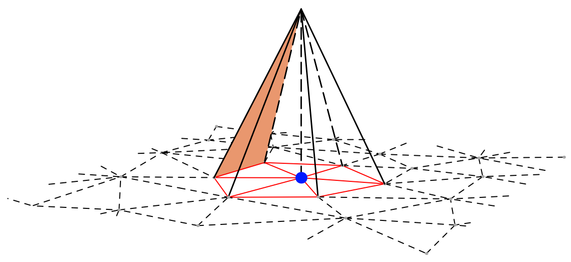


Figure 6.2: A triangle mesh with linear shape functions. At the blue node the shape functions are at unity height and have only local support in the red elements.

Shape functions have only local support[6] in Ω_e. This has two consequences for the weighted residual method. One, the contribution to the global matrix needs to be computed if and only if the shape function and the weighting function are in the same element. Second, by requiring only local support for the An-

[6]In mathematics the support of a function is the range in which it is non-zero.

satzfunktionen, matrix A from the above example becomes sparse. Also, shape functions retain the feature of being nicely differentiable.

What follows is a list of shape functions [7], where r, s, t always refer to local coordinates. The presented shape functions are C^0 continuous and of the serendipity[7] family. The position of the nodes supporting the shape functions in the mother elements can be seen from Figures 6.4 to 6.6. A list of shape functions is compiled in Table 6.1[8].

Table 6.1: Shape functions for line elements in $\mathbb{R}^1$ in the unit domain $[-1, 1]$ and in $\mathbb{R}^2$ for quads in the unit domain $[-1, 1]^2$ and for triangles in the unit domain $[0, 1] \times [0, 1 - r]$ of order 1 and 2.

order	line	quad	triangle
1	$1/2\,(1-r)$	$1/4\,(r+1)\,(s+1)$	$-r-s+1$
	$1/2\,(r+1)$	$1/4\,(1-r)\,(s+1)$	r
		$1/4\,(1-r)\,(1-s)$	s
		$1/4\,(r+1)\,(1-s)$	
2	$1/2\,(r^2-r)$	$1/4\,(r+1)\,(s+1)\,(r+s-1)$	$2r^2+4sr-3r+2s^2-3s+1$
	$1-r^2$	$1/4\,(1-r)\,(s+1)\,(-r+s-1)$	$2r^2-r$
	$1/2\,(r^2+r)$	$1/4\,(1-r)\,(1-s)\,(-r-s-1)$	$2s^2-s$
		$1/4\,(r+1)\,(1-s)\,(r-s-1)$	$-4r^2-4sr+4r$
		$1/2\,(1-r^2)\,(s+1)$	$4rs$
		$1/2\,(1-r)\,(1-s^2)$	$-4s^2-4rs+4s$
		$1/2\,(1-r^2)\,(1-s)$	
		$1/2\,(r+1)\,(1-s^2)$	

6.3.2 Operators

In this section we derive the discretised versions of the partial differential equation operators presented in Equations 4.1 to 4.7. At the end of a derivation we will highlight the implemented formula.

6.3.2.1 Diffusion

We start from the weighted residual formulation

$$\int_\Omega r\, w_i\, d\Omega = 0$$

[7]The implications of the fact that these element are serendipity elements will be introduced at a later stage and is of no relevance at this point.

[8]For $\mathbb{R}^3$ shape functions consult the program file: Imtek‘ShapeFunctions.m. See the implementation part in Chapter 7 for more details.

with the residual

$$r = \mathcal{L}(u) - f.$$

Inserting the diffusion operator from Equation 4.1 we obtain

$$\mathcal{L}(u) = \nabla^T \cdot (-\sigma \cdot \nabla u) \text{ and } f = 0$$

and inserting into the weighted residual integral we obtain

$$\int_\Omega \nabla^T \cdot (-\sigma \cdot \nabla u)\; w_i \, d\Omega = 0. \tag{6.5}$$

If shape functions were inserted now, they would need to be of at least second order. A linear shape function would simply vanish by differentiating twice. To circumvent this shortcoming the residual r_Ω is chosen in a weak sense. That is we apply the first of Green's Theorems[9]

$$\int_\Omega \nabla^T \cdot (k \cdot \nabla g)\, h \; d\Omega = -\int_\Omega \left((k \cdot \nabla g)^T \cdot \nabla h \right) \; d\Omega + \int_{\partial\Omega} h \, (k \cdot \nabla g) \cdot n \; d\partial\Omega, \tag{6.6}$$

where g and h are arbitrary differentiable functions defined on Ω and its boundary $\partial\Omega$ and k is a second order tensor. Note that the weak form reduces the demand on the differentiability of the differential equation by one but increases the demand on differentiability of the weighting function by one[10].

Equation 6.6 applied to Equation 6.5 results in

$$\int_\Omega (\sigma \cdot \nabla u)^T \cdot \nabla w_i \, d\Omega - \int_{\partial\Omega} \left((\sigma \cdot \nabla u)^T \cdot \mathbf{n} \right) w_i \, d\partial\Omega = 0$$

with

$$u = u_{\partial\Omega}(\mathbf{x}, t) + \sum_{j=1}^{N} u_j(t) \alpha_j(\mathbf{x}, t)$$

and

$$w_i = \alpha_i.$$

We thus have a system of linear equations

$$\sum_{j=1}^{N} \left[\left(\int_\Omega (\sigma \cdot \nabla \alpha_j)^T \cdot \nabla \alpha_i \, d\Omega \right) u_j + \underbrace{\int_\Omega (\sigma \cdot \nabla u_{\partial\Omega})^T \cdot \nabla \alpha_i \, d\Omega}_{\text{Dirichelt Boundary}} - \underbrace{\int_{\partial\Omega} (\sigma \cdot \nabla \alpha_j)^T \cdot \mathbf{n} \, \alpha_j d\partial\Omega}_{\text{Neumann Boundary}} \right] = 0 \tag{6.7}$$

The integrals and the Ansatzfunktionen are defined and valid on the entire do-

[9]This is sometimes also referred to as the Green-Gauss Theorem or the Green Lemma.
[10]Some authors already call Equation 6.3 weak form.

main. This brings considerable computational cost. In order to reduce this cost we replace the Ansatzfunktionen α with the shape functions N which have only local support[11]. Only in regions where the support of N_i overlaps the support of N_j the integral has a result different from 0 and thus an element point of view is appropriate

$$\boxed{S_{ij}^{(e_k)} = \int_{\Omega^{(e_k)}} (\sigma \cdot \nabla N_j)^T \cdot \nabla N_i \, d\Omega}. \tag{6.8}$$

The operator's contribution in the sense defined by Equation 2.2 is thus

$$\mathcal{L}_{\text{FEMDiffusion}}(e_k) \rightarrow \{\mathbf{0}^{(e_k)}, S_{ij}^{(e_k)}, \mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}\}$$

6.3.2.2 Convection

The convection operator as defined by Equation 4.4 is

$$\beta \cdot \nabla u.$$

We replace ∇u with the shape function derivative ∇N_j and multiply with the shape function N_i as a test function to get

$$\boxed{S_{ij}^{(e_k)} = \int_{\Omega^{(e_k)}} (\beta \cdot \nabla N_j)^T \cdot N_i \, d\Omega}. \tag{6.9}$$

The operator's contribution to the system is

$$\mathcal{L}_{\text{FEMConvection}}(e_k) \rightarrow \{\mathbf{0}^{(e_k)}, S_{ij}^{(e_k)}, \mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}\}.$$

6.3.2.3 Reaction

From the reaction operator defined by Equation 4.3 as

$$au$$

by replacing u with the shape function N_j and multiplying with the shape function N_i as test function we get

$$\boxed{S_{ij}^{(e_k)} = \int_{\Omega^{(e_k)}} (aN_j)^T \cdot N_i \, d\Omega}. \tag{6.10}$$

[11]Apart from the support Ansatzfunktionen and shape functions are the same.

The contribution to the system of equations is

$$\mathcal{L}_{\text{FEMReaction}}(e_k) \rightarrow \{\mathbf{0}^{(e_k)}, S_{ij}^{(e_k)}, \mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}\}.$$

6.3.2.4 Load

From the load operator defined by Equation 4.2 as

$$L$$

multiplying with the test function N_i yields

$$\boxed{L_i^{(e_k)} = \int_{\Omega^{(e_k)}} f N_i \, d\Omega}. \tag{6.11}$$

The contribution to the system of equations is

$$\mathcal{L}_{\text{FEMLoad}}(e_k) \rightarrow \{L_i^{(e_k)}, \mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}\}.$$

6.3.2.5 Operator for first order time derivative

We start from the integral defined by Equation 4.6

$$\int_\Omega \tau_1 \frac{\partial u}{\partial t} \, d\Omega$$

with

$$u = u_{\partial\Omega}(\mathbf{x}, t) + \sum_{j=1}^{N} u_j(t) N_j(\mathbf{x}) \text{ and } w_i = N_i.$$

Since the shape function N_j and the test function N_i are not time dependent we may write [9]

$$\sum_{j=1}^{N} \left(\int_\Omega \tau_1 \frac{\partial N_j}{\partial t} N_i \, d\Omega \right) u_j = \frac{\partial}{\partial t} \sum_{j=1}^{N} \left(\int_\Omega \tau_1 N_j N_i \, d\Omega \right) u_j$$

which leads to

$$\boxed{D_{ij}^{(e_k)} = \int_{\Omega^{(e_k)}} (\tau_1 N_j)^T \cdot N_i \, d\Omega}. \tag{6.12}$$

The contribution to the system of equations is

$$\mathcal{L}_{\text{FEMTransient}}(e_k) \rightarrow \{\mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}, D_{ij}^{(e_k)}, \mathbf{0}^{(e_k)}\}.$$

Remark: Note that this is the same as the reaction operator. Strictly speaking we do not need the transient operators.

6.3.2.6 Operator for second order time derivative

Along the same line of reasoning as for the first order time derivative operator we may write for the second order time derivative operator defined by Equation 4.7

$$\boxed{M_{ij}^{(e_k)} = \int_{\Omega^{(e_k)}} (\tau_2 N_j)^T \cdot N_i \, d\Omega} \tag{6.13}$$

and thus

$$\mathcal{L}_{\text{FEMTransient}}(e_k) \rightarrow \{\mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}, \mathbf{0}^{(e_k)}, M_{ij}^{(e_k)}\}.$$

6.3.2.7 Generalisation

It is well worth noting that the operators can be generalised further to one master operator. In fact the generalisation is then done in several directions. We start by introducing

$$\mathcal{D}\left(N, \mathbf{r}, o\right)$$

where $\mathcal{D}$ is the general derivative operator. $\mathcal{D}$ differentiates the shape function N with respect to the local coordinate vector $\mathbf{r}$ up to order o. The order ranges from $0 \ldots n$. Where $\mathcal{D}(N, \mathbf{r}, 0) = N$. The general partial differential operator is then

$$\int_{\Omega^{(e_k)}} \coprod_{i=1}^{p} \kappa_i \otimes_i \mathcal{D}\left(N_i, \mathbf{r}, o_i\right) \, d\Omega^{(e_k)}.$$

The integral has the same function as before. κ_i is the i^{th} input function multiplied in an appropriate manner $\otimes_i$ with the derivative operator $\mathcal{D}$. Depending on the order o_i of the derivative of the shape function the multiplication is a normal $\times$ or a dot product $\cdot$, where the dot-product operator is defined as

$$\coprod_{i=1}^{p} x_i = x_p \cdot \ldots \cdot x_1.$$

Here the transposes are implied by the multiplication.

Example: The diffusion operator is then

$$\int_{\Omega^{(e_k)}} \coprod_{i=1}^{2} \kappa_i \otimes_i \mathcal{D}\left(N_i, \mathbf{r}, o_i\right) \, d\Omega^{(e_k)} =$$
$$\int_{\Omega^{(e_k)}} \kappa_2 \otimes_2 \mathcal{D}\left(N_2, \mathbf{r}, o_2\right) \cdot \kappa_1 \otimes_1 \mathcal{D}\left(N_1, \mathbf{r}, o_1\right) \, d\Omega^{(e_k)}.$$

with

$$\kappa_2 = \sigma \text{ and } \kappa_1 = 1$$

and

$$o_2 = o_1 = 1$$

which leads to

$$\int_{\Omega^{(e_k)}} \sigma \otimes_2 \mathcal{D}\left(N_2, \mathbf{r}, 1\right) \cdot 1 \otimes_1 \mathcal{D}\left(N_1, \mathbf{r}, 1\right) \, d\Omega^{(e_k)} \quad = \quad \int_{\Omega^{(e_k)}} \sigma \cdot \nabla N_2 \cdot 1 \times \nabla N_1 \, d\Omega^{(e_k)}$$

6.3.3 Numerical integration in unit domains

The operators need to be integrated over the domain of an element $\Omega^{(e_k)}$. This element domain can be located anywhere in the simulation domain Ω. To simplify matters we proceed as follows. Each element is mapped to a unit domain mother element. After the mapping the integration is performed over this mother element. To this end the integral itself is replaced by a finite summation where we relay on integration points $\mathbf{r}_i$ and integration weights $\hat{c}_i$ to reach a desired accuracy.

For the mother line element, which is defined in $\Omega^{(M)} \in [-1, 1]$, we use Gauß-Legendre integration [7]

$$\int_{-1}^{1} F(r) \, dr = \sum_{i=1}^{q} c_i F(r_i),$$

where F is the integral kernel evaluated at the integration points r_i and weighted with the integration weights c_i. q is the number of integration points.

For the mother quad element, which is defined in $[-1, 1]^{d=2}$, we use

$$\int_{-1}^{1} \int_{-1}^{1} F(r, s) \, dr \, ds = \sum_{i=1}^{q} \sum_{j=1}^{q} c_i c_j F(r_i, s_j)$$

and the same procedure is applied. To simplify matters further we can rewrite the double summation to a single summation by adjusting the summation length and the product factors to

$$\sum_{i=1}^{q} \sum_{j=1}^{q} c_i c_j F(r_i, s_j) = \sum_{i=1}^{n} \hat{c}_i F(\mathbf{r}_i), \tag{6.14}$$

where $n = q^d$ is the number of integration points to the power of the space dimension d. For $d = 1$ we have $n = q$ and for $d = 2$ we have q^2 integration points. The $\hat{c}_i$ are the coefficients and $\mathbf{r}_i$ is a vector which holds the r_i and s_j appropriately adjusted. Please note that this formulation also holds for $d = 1$. Then $c_i = \hat{c}_i$, $n = q^1$ and $r_i = \mathbf{r}_i$.

The theory extends along similar lines for hexahedron mother elements and generally for n-dimensional hypercubes.

For triangular mother elements and tetrahedra mother elements we also use Equation 6.14. Here, however, the integration limits are different, namely

$$\int_0^1 \int_0^{1-r} F(r,s)\,dr\,ds = \sum_{i=1}^{n} \hat{c}_i F(\mathbf{r}_i)$$

and

$$\int_0^1 \int_0^{1-r} \int_0^{1-s} F(r,s,t)\,dr\,ds\,dt = \sum_{i=1}^{n} \hat{c}_i F(\mathbf{r}_i).$$

The integration points and weights for different mother elements are derived in the following sub-sections.

6.3.3.1 Line, quad & hexahedron elements

We start with line elements and thus the dimension is $d = 1$. The integration points r_i for elements based on the domain $\Omega \in [-1,1]$ are the roots of the Legendre-Polynomial $P_z(x)$ which is defined [12] by

$$P_z(x) = \frac{1}{2^z z!} \frac{d^z}{dx^z} \left(x^2 - 1\right)^z \text{ with } z \in \mathbb{N}_0.$$

The Legendre-Polynomial $P_z(x)$, $z \geq 1$ has z single roots in the open interval $(-1,1)$ (for proof see [12]). First we compute the z roots from the recursion formula

$$\begin{aligned} P_0(x) =& 1 \\ P_1(x) =& x \\ P_z(x) =& \left(\frac{2z-1}{z}\right) x\, P_{z-1}(x) - \left(\frac{z-1}{z}\right) P_{z-2}(x) \text{ for } z \geq 2 \end{aligned}$$

and then compute the integration weights c_i by comparing coefficients. This is illustrated in the following Example [12].

Example: To compute 3 integration points on a unit line element we use the Legendre-Polynomial of degree $z = 3$

$$P_3(x) = \frac{5}{2}x^2 - \frac{3}{2}x$$

which has integration points r_i (roots) $i = 1,..,z$

$$x_1 = -\sqrt{\tfrac{3}{5}}, \quad x_2 = 0, \quad x_3 = \sqrt{\tfrac{3}{5}} \ .$$

To find the integration weights c_i we calculate

$$\int_{-1}^{1} f(x)dx \approx c_1 f\left(-\sqrt{\frac{3}{5}}\right) + c_2 f\left(0\right) + c_3 f\left(\sqrt{\frac{3}{5}}\right),$$

which is exact for polynomials of degree $m \leq 5$. We now evaluate the integrals for $f(x) = x^{i-1}$ where $i = 1, .., z$

$$\begin{array}{rclcl} f(x) = 1 & : & \int_{-1}^{1} 1\, dx = 2 & = & c_1 + c_2 + c_3 \\ f(x) = x & : & \int_{-1}^{1} x\, dx = 0 & = & -\sqrt{\frac{3}{5}}c_1 + \sqrt{\frac{3}{5}}c_3 \\ f(x) = x^2 & : & \int_{-1}^{1} x^2\, dx = \frac{2}{3} & = & \frac{3}{5}c_1 + \frac{3}{5}c_3 \end{array}.$$

We thus have

$$\int_{-1}^{1} f(x)dx = \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f\left(0\right) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right) + E$$

where E is an error.

This process can be automated and implemented and will result in analytical integration points and weights which are important for a symbolic finite element environment. We present a table of integration weights and points. Note that for $r_i > 5$ the roots become complex and other numerical schemes have to be applied.

To proceed for quads, hexahedra or higher dimensional hypercubes we take the outer product of the integration points and weight to obtain the new vector integration points and integration weight. For example the integration points and weights for a quad with $q = 2$ integration points results in $n = 2^2 = 4$ and

$$\hat{c}_i = \{1, 1, 1, 1\}$$

and

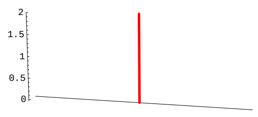
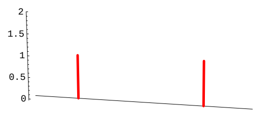
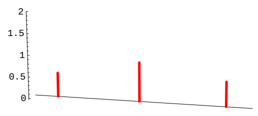
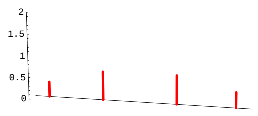
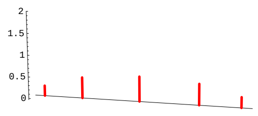
$$\mathbf{r}_i = \{\{\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\}, \{\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\}, \{-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\}, \{-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\}\}.$$

6.3.3.2 Triangle & tetrahedra elements

We give the following tables: Table 6.3 for triangles and Table 6.4 for tetrahedra integration points and weights.

Remark: Some authors [7] give the integration weights for triangles and tetrahedra in a slightly different form. This is due to the fact that the actual integration formula used by those authors is different. For example for

Table 6.2: Analytical integration points and weights for line elements in the unit element $\Omega \in [-1, 1]$ in $\mathbb{R}^1$. In those integration point formulae where a $\pm$ sign appears the convention in this text is to first use the positive and then the negative values

position and magnitude of integration points and weight	order $2n-1$	r_i	$\hat{c}_i$
	1	0	2
	3	$\pm\frac{1}{3}\sqrt{3}$	1 1
	5	$\pm\frac{1}{5}\sqrt{15}$ 0	$\frac{5}{9}$ $\frac{8}{9}$
	7	$\pm\sqrt{\frac{3}{7}+\frac{1}{21}\sqrt{30}}$ $\pm\sqrt{\frac{1}{35}\left(15-2\sqrt{30}\right)}$	$\frac{1}{2}-\frac{1}{36}\sqrt{30}$ $\frac{1}{36}\left(18+\sqrt{30}\right)$
	9	$\pm\sqrt{\frac{5}{9}+\frac{2}{63}\sqrt{70}}$ $\pm\frac{1}{3}\sqrt{\frac{1}{7}\left(35-2\sqrt{70}\right)}$ 0	$\frac{1}{900}\left(322\pm 13\sqrt{70}\right)$ 0 $\frac{1}{900}\left(322\pm 13\sqrt{70}\right)$

triangles

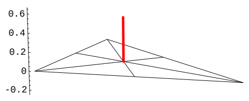
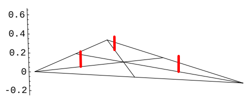
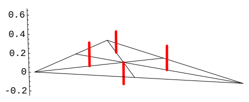
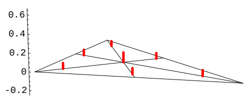
$$\int_0^1 \int_0^1 F(r,s)\,dr\,ds = \frac{1}{2}\sum_{i=1}^{n} \hat{c}_i F(\mathbf{r}_i)$$

and for tetrahedra

$$\int_0^1 \int_0^1 \int_0^1 F(r,s,t)\,dr\,ds\,dt = \frac{1}{6}\sum_{i=1}^{n} \hat{c}_i F(\mathbf{r}_i)$$

is used. In our case the factors $1/2$ and $1/6$ are multiplied into the integration weights and we thus can use Equation 6.14 for all types of elements in 1 or arbitrarily more dimensions.

Table 6.3: Triangle integration points and weights

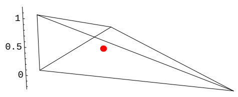
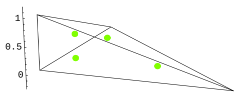
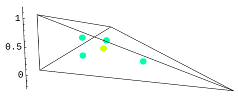
position and magnitude of integration points and weight	order	$\mathbf{r}_i$	$\hat{c}_i$
	1	$\left\{\frac{1}{3},\frac{1}{3}\right\}$	$\frac{1}{2}$
	2	$\left\{\frac{1}{6},\frac{1}{6}\right\}$ $\left\{\frac{2}{3},\frac{1}{6}\right\}$ $\left\{\frac{1}{6},\frac{2}{3}\right\}$	$\frac{1}{6}$ $\frac{1}{6}$ $\frac{1}{6}$
	3	$\left\{\frac{1}{3},\frac{1}{3}\right\}$ $\left\{\frac{3}{5},\frac{1}{5}\right\}$ $\left\{\frac{1}{5},\frac{3}{5}\right\}$ $\left\{\frac{1}{5},\frac{1}{5}\right\}$	$-\frac{9}{32}$ $\frac{25}{96}$ $\frac{25}{96}$ $\frac{25}{96}$
	5	$\frac{1}{21}\left\{\begin{array}{c}\{6-\sqrt{15},6-\sqrt{15}\}\\\{9+2\sqrt{15},6-\sqrt{15}\}\\\{6-\sqrt{15},9+2\sqrt{15}\}\\\{6+\sqrt{15},9-2\sqrt{15}\}\\\{6+\sqrt{15},6+\sqrt{15}\}\\\{9-2\sqrt{15},6+\sqrt{15}\}\\\{7,7\}\end{array}\right\}$	$\frac{(155-\sqrt{15})}{2400}$ $\frac{(155-\sqrt{15})}{2400}$ $\frac{(155-\sqrt{15})}{2400}$ $\frac{(155+\sqrt{15})}{2400}$ $\frac{(155+\sqrt{15})}{2400}$ $\frac{(155+\sqrt{15})}{2400}$ $\frac{9}{80}$

6.3.3.3 Element mapping

For the integration the question remains as to how to map an arbitrary element to its respective mother element. Also of interest is a methodology to evaluate an operator's input function in the un-mapped element.

We start with the second question. To map from the mother element in r, s, t

Table 6.4: Tetrahedra integration points and weights

position and magnitude of integration points and weight	order	$\mathbf{r}_i$	$\hat{c}_i$
	1	$\frac{1}{4}\{\{1,1,1\}\}$	$\frac{1}{6}$
	2	$\frac{1}{20}\left\{\begin{array}{c}\{5+3\sqrt{5},5-\sqrt{5},5-\sqrt{5}\}\\\{5-\sqrt{5},5+3\sqrt{5},5-\sqrt{5}\}\\\{5-\sqrt{5},5-\sqrt{5},5+3\sqrt{5}\}\\\{5-\sqrt{5},5-\sqrt{5},5-\sqrt{5}\}\end{array}\right\}$	$\begin{array}{c}\frac{1}{24}\\\frac{1}{24}\\\frac{1}{24}\\\frac{1}{24}\end{array}$
	3	$\frac{1}{12}\left\{\begin{array}{c}\{3,3,3\}\\\{6,2,2\}\\\{2,6,2\}\\\{2,2,6\}\\\{2,2,2\}\end{array}\right\}$	$\begin{array}{c}-\frac{4}{30}\\\frac{9}{120}\\\frac{9}{120}\\\frac{9}{120}\\\frac{9}{120}\end{array}$

coordinates to the element in x, y, z coordinates we write [13]

$$x = N_1'x_1 + N_2'x_2 + ... = \mathbf{N}' \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix} = \mathbf{N}'\mathbf{x}$$

$$y = N_1'y_1 + N_2'y_2 + ... = \mathbf{N}' \begin{pmatrix} y_1 \\ y_2 \\ \vdots \end{pmatrix} = \mathbf{N}'\mathbf{y} \tag{6.15}$$

$$z = N_1'z_1 + N_2'z_2 + ... = \mathbf{N}' \begin{pmatrix} z_1 \\ z_2 \\ \vdots \end{pmatrix} = \mathbf{N}'\mathbf{z},$$

where $\mathbf{N}'$ are standard C_0 shape functions in local r, s, t coordinates. The x_1, y_1, z_1 are the element nodal points in global coordinates. This is illustrated by an example.

Example: To find the mapping from a mother triangle element to a triangle in global domain we proceed as follows: The local shape functions are

$$\{1 - r - s, r, s\}$$

The global triangle is supposed to be at coordinates

$$\{\{1,1\},\{2,1\},\{3/2,3/2\}\}.$$

The mapping then is

$$\{\{1-r-s,r,s\}\}\cdot\{\{1,1\},\{2,1\},\{3/2,3/2\}\}=\{1+r+s/2,1+s/2\}.$$

By inspection, for $\{r,s\}=\{0,0\}$ the mapping results in $\{1,1\}$.

In this case we used linear shape functions for the mapping. If for computation also linear mapping is employed, which implies

$$\mathbf{N}=\mathbf{N}'$$

we speak of isoparametric elements. In the case where a higher order shape function is used than the order of the computational shape function we speak of superparametric elements and in the case where the order of the computation shape function is higher than the order of the mapping shape functions we speak of subparametric elements. Even though other settings are possible, in this work we use isoparametric elements.

To answer the question of how to map an arbitrary element to its respective mother element, we note that in the expression in Equations 4.1 to 4.7 the derivatives of the shape functions also need to be computed. If we consider local coordinates r,s,t and apply the rules of partial differentiation we may write

$$\begin{pmatrix} \frac{\partial N_i}{\partial r} \\ \frac{\partial N_i}{\partial s} \\ \frac{\partial N_i}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{pmatrix} \begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{pmatrix} = \mathbf{J} \begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{pmatrix},$$

where $\mathbf{J}$ is the Jacobian matrix. We note that the left-hand side is known since the N_i are local shape functions. The coordinates x,y,z are known through Equation 6.15 and can be written in terms of the mapping shape functions $\mathbf{N}'$ as

$$\mathbf{J} = \begin{pmatrix} \sum \frac{\partial N_i'}{\partial r} x_i & \sum \frac{\partial N_i'}{\partial r} y_i & \sum \frac{\partial N_i'}{\partial r} z_i \\ \sum \frac{\partial N_i'}{\partial s} x_i & \sum \frac{\partial N_i'}{\partial s} y_i & \sum \frac{\partial N_i'}{\partial s} z_i \\ \sum \frac{\partial N_i'}{\partial t} x_i & \sum \frac{\partial N_i'}{\partial t} y_i & \sum \frac{\partial N_i'}{\partial t} z_i \end{pmatrix} = \begin{pmatrix} \frac{\partial N_1'}{\partial r} & \frac{\partial N_2'}{\partial r} & \dots \\ \frac{\partial N_1'}{\partial s} & \frac{\partial N_2'}{\partial s} & \dots \\ \frac{\partial N_1'}{\partial t} & \frac{\partial N_2'}{\partial t} & \dots \end{pmatrix} \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \end{pmatrix}, \tag{6.16}$$

where the right-hand side is readily available, i.e. the mapping shape function derivatives and the element coordinates.

For the computation we need the derivatives of the global shape functions which can now be expressed as

$$\begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{pmatrix} = \mathbf{J}^{-1} \begin{pmatrix} \frac{\partial N_i}{\partial r} \\ \frac{\partial N_i}{\partial s} \\ \frac{\partial N_i}{\partial t} \end{pmatrix}$$

One further point has to be considered when employing element mapping. The element integration in Equations 4.1 to 4.7 is defined in global coordinates. To transform to local coordinates an adjustment has to be made

$$dx\, dy\, dz = \det(\mathbf{J}) dr\, ds\, dt$$

6.3.3.4 Pre-integrated shape functions

In our case the integral kernel $F(\mathbf{r}_i)$ involves the shape functions and their derivatives. Since the integration is done in the unit mother element the integration points remain the same. We can thus pre-integrate the shape functions and their derivatives at the integration points $\mathbf{r}_i$. In the actual integration process we then only need look-up tables to look up the pre-integrated shape function or derivative and multiply them appropriately with the integrated operator's input functions and integration weights. Some authors go further [8] and thus establish pre-evaluated element matrices which then only need to be multiplied with the Jacobian. This, however, rules out the use of input functions and is thus of limited interest only.

The disadvantage is then that the number of integration points and weights is in some way attached to the shape function. It is then a little more difficult to change the integration order of an element. In other words a "new" element with the same shape function but different integration order has to be pre-integrated in order to use an element with a different integration order.

6.3.3.5 Input function integration

The integration of the operator's input function can be done in the following manner: we make use of the fact that we have the pre-integrated shape functions. We dot-multiply them with the elements' coordinates and evaluate the input function at these points. The appropriate weighting can then be done in the integration procedure. Please see the implementation in Chapter 7.4.3 on page 104 for further detail.

6.3.4 Area coordinates

In previous sections we propagated the following procedure for the finite element method:

1. Map the mesh element to its corresponding unit element
2. Multiply the input function to the mapped element
3. Integrate appropriately over the unit element.

There is, however, a further method for triangle and tetrahedra elements to compute the finite elements. This method relies on so-called area coordinates[12]. Using area coordinates has the advantage that analytical results for the element integration exist in some circumstances. As long as the operator's input functions are not dependent on the coordinates, an analytical result of the integral exists [6, 13]. This circumstance makes the evaluation very fast. However, such results exist only for triangles and tetrahedra. Even though it is possible to use higher order elements it is not possible to use serendipity type elements or quad or hexahedron elements.

Nevertheless, area coordinates are of importance for interpolation purposes and there is an efficient way to check if coordinates are inside an element or not. Both functionalities are essential in this work and thus we take the time to introduce area coordinates.

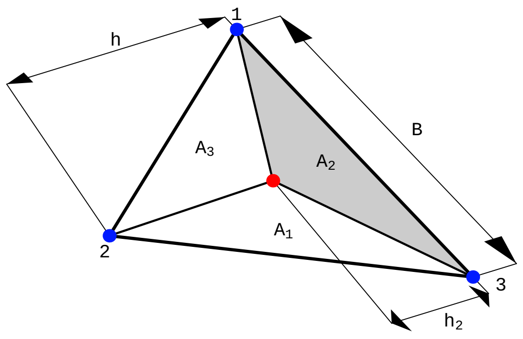


Figure 6.3: Area coordinates relate the area A_2 to the total area of the triangle [6].

Area coordinates are normalised coordinates for triangles and tetrahedra [6]. We consider the triangle in Figure 6.3. We define

$$L_2 = \frac{h_2}{h}$$

and L_1 and L_3 similarly. We note that

$$A_2 = \frac{Bh_2}{2}$$

[12]Area coordinates are also called barycentric coordinates

and that the entire area of the triangle is

$$A = \frac{Bh}{2}$$

which leads to

$$L_2 = \frac{Bh_2/2}{Bh/2} = \frac{A_2}{A}$$

or generally

$$L_i = \frac{A_i}{A}.$$

Area coordinates are defined by relating the area of sub-triangles A_i in terms of the complete area A of the triangle. A_i may be expressed as

$$A_i = \frac{1}{2}\begin{vmatrix} 1 & x_p & y_p \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} = \frac{1}{2}\left(a_i + b_i x_p + c_i y_p\right)$$

where x_p and y_p are the coordinates of an arbitrary point inside the triangle and

$$\begin{aligned} a_i &= x_j y_k - x_k y_j \\ b_i &= y_j - y_k \\ c_i &= x_k - x_j. \end{aligned}$$

Area coordinates can be used as shape functions if we consider for example that $L_2 = 1$ if p is at node 2 and $L_2 = 0$ if p is at node 1 or 3. Furthermore, the variation in between is linear. In this case the shape functions are

$$N_i = L_i, \text{ with } i = 1, 2, 3.$$

The argumentation for tetrahedra is along similar lines. To construct elements of higher order we list without derivation for quadratic triangle elements [3]

$$N_i = L_i(2L_i - 1) \text{ with } i = 1, 2, 3$$

and

$$N_4 = 4L_1L_2,\ N_5 = 4L_2L_3,\ N_6 = L_3L_1.$$

We would like to point out that the exact integration formulae exist

$$\int_{\Omega^{(e_k)}} L_1^a L_2^b L_3^c \, d\Omega^{(e_k)} = \frac{a!b!c!}{(a+b+c+2)!} 2A,$$

with A the triangles area and $a, b, c \in \mathbb{I}$. Also

$$\int_{\partial\Omega^{(e_k)}} L_1^a L_2^b \, \partial\Omega = \frac{a!b!}{(a+b+1)!} S,$$

where S is the path length. For $\mathbb{R}^3$ we have

$$\int_{\Omega^{(e_k)}} L_1^a L_2^b L_3^c L_4^d \, d\Omega^{(e_k)} = \frac{a!b!c!d!}{(a+b+c+d+3)!} 6V,$$

where V is the tetrahedron's volume.
To compute if a coordinate x_p is inside a linear tetrahedron[13] we use [5]

$$\min(L_i, 1 - L_i) \geq 0$$

where

$$\begin{pmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

for $\mathbb{R}^3$.

Remark: Care has to be taken when handling computational precision[14].

Remark: For higher order triangles and tetrahedra we simply use the primary nodes. The notion of primary and secondary nodes will be properly introduced in Chapter 6.3.5. This simplification may be wrong for higher order serendipity elements and improvement is needed for such cases. For linear quads and hexahedra we split them into triangles and tetrahedra respectively. For higher order quads and hexahedra we need to insert additional points if we would like to use the splitting method. We have not checked as to how the splitting may influence the interpolation results.

6.3.5 Meshes

In the following section we introduce unstructured meshes and their elements.

6.3.5.1 Mesh elements

A list of mesh elements and their primary and secondary incidents are depicted in Figures 6.4 to 6.6.

[13]To check if a coordinate is inside a triangle we can proceed in a similar fashion.

[14]The single most important rule is to never check against zero. A statement of the form $p = 0$ should always be recast into the following form: $|p| \leq \epsilon$, where ϵ is a small number.

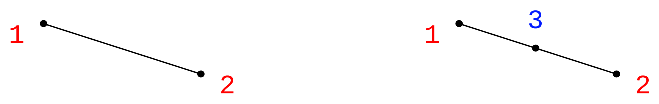


Figure 6.4: 1D mesh elements. Red nodes are primary nodes and blue nodes are secondary nodes. Left: Linear mesh elements. Right: Second order mesh elements.

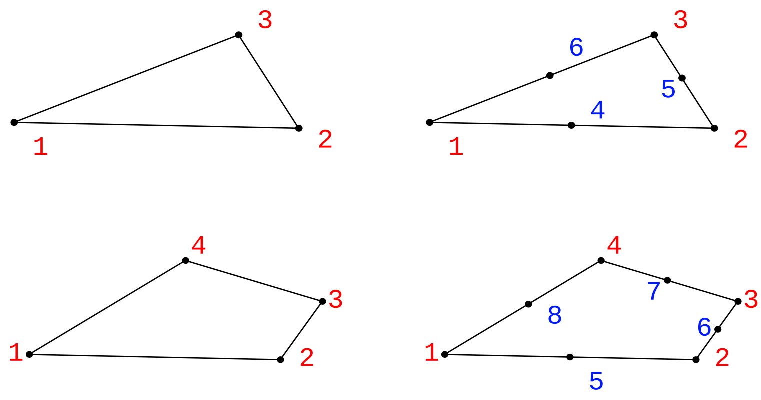


Figure 6.5: 2D mesh elements. Red nodes are primary nodes and blue nodes are secondary nodes. Left: Linear mesh elements. Right: Second order mesh elements.

Here the so-called primary nodes are coloured in red and the secondary nodes are in blue. The primary nodes are sufficient to plot the element and they are also sufficient for numerous computations. If an element is of higher order then it has additional secondary nodes, for example to support higher order shape functions. The secondary nodes supplement the primary nodes. This means an element made up of primary nodes such as the triangle

$$\{n_1, n_2, n_3\} \in e_1$$

can be converted to a higher order element by using

$$\{n_1, n_2, n_3, n_4, n_5, n_6\} \in e_1.$$

It is important to note that the nodes n_1 to n_3 are the same nodes and they are

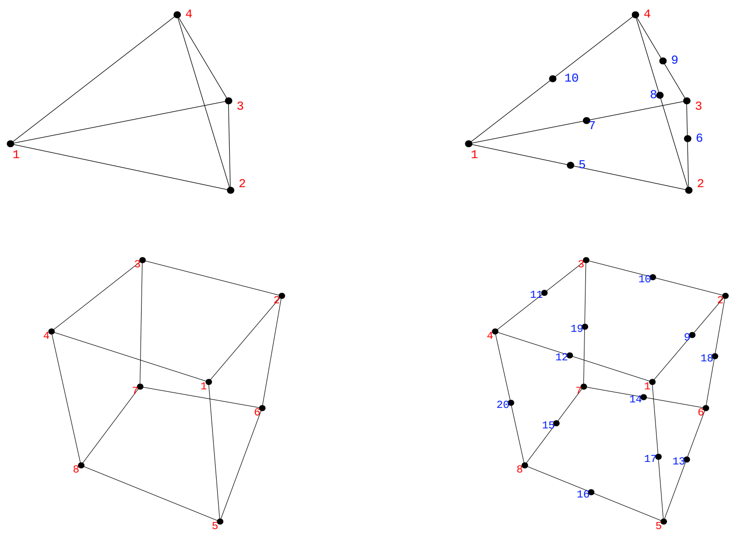


Figure 6.6: 3D mesh elements. Red nodes are primary nodes and blue nodes are secondary nodes. Left: Linear mesh elements. Right: Second order mesh elements.

at the same position in the incidents list,

$$i_1 = \{n_1, n_2, n_3\}.$$

These are the primary nodes since they are present in all higher order types of the specific element. The additional nodes n_4 to n_6 have to be added and they are appended to the incidents list i_1. These nodes are called secondary nodes.

6.3.5.2 Serendipity type elements

Serendipity[15] type elements are truncated polynomials. The shape functions of higher than first order presented in this work are serendipity elements. Since these elements do not need points inside the elements - nodes on the element boundary suffice - the resulting element matrices are smaller. Furthermore, these elements are suitable for curved linear boundaries as well. Figures 6.4 to 6.6 do not display the serendipity character of the implemented mesh elements.

6.3.6 Numerical challenges

In this section some specific problems are addressed. Some of the problems are shortcomings of the finite element method while others are general problems

[15]Serendipity elements are named after the fairytale "The three princes from Serendip." by Horace Walpole. These princes had the ability to make unexpected and fortunate discoveries and thus the word serendipity was coined [7].

of the weighted residual method or some are problems of the level set method. They are presented in no particular order.

6.3.6.1 Recovery of derivatives

We use the interpolation with area coordinates and recover the derivatives via those. The interpolation function is then input to the operator's input function.

6.3.6.2 Re-initialisation

To avoid that the level set becomes too flat it is important that the level set remains a signed distance function. However, while updating the level set with Equation 5.3 and at the same time satisfying $\nabla u = 0$ from the Navier-Stokes equations 5.1, it will not remain as such [10]. To counteract the deviation of the signed distance function from $|\nabla \phi| = 1$, the computation is interrupted periodically and the level set ϕ is re-initialised.

To re-initialise ϕ two methods are common. The first possible choice is to solve the following equation

$$\frac{\partial}{\partial t'} + F|\nabla \phi| = 0,$$

where F is the normal velocity. We note that t' is a pseudo time variable. To find the new signed distance we set $F = 1$ and follow the interface forward and backward in time t' until each node has had a change in sign. The crossing times are the new signed distances. This introduces another non-linear equation.

We have chosen a second, simpler alternative. In this case the ϕ_0 is found, for example by a contour plotting routine, and then the signed distance from each node is recalculated by computing the distance to the interface.

In both methods we can take advantage of the fact that we need not re-initialise the entire domain. It is sufficient to re-initialise a so-called narrow band around ϕ_0 since only the ϕ_0 interface is of interest. Should ϕ_0 move then the appropriate nodes are re-initialised. It would be desirable to avoid the re-initialisation step entirely. In Figure 6.7 we display a non-signed distance function ϕ_2. Both, the entire re-initialisation and the narrow band re-initialisation are shown. In the entire re-initialisation a new ϕ_1 is found in the entire domain. In the narrow band, which is represented by the elements marked in red, only elements in some vicinity of the interface $\Gamma \in \phi_2$ are re-initialised.

6.3.6.3 Level-Set correction

Re-initialisation is not done without changing the zero level-set ϕ_0. In Figure 6.8 two types or errors are discussed. Both errors lead to a loss of mass in the level set method. This is explained in Figure 6.9.

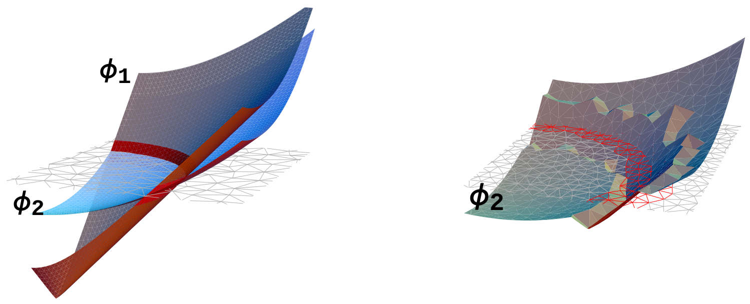


Figure 6.7: Left: Once a level set function has deviated from the signed distance property as ϕ_2 a re-initialisation step has to be made. In a full re-initialisation each mesh node is assigned the newly computed value of ϕ_1. Right: A narrow band re-initialisation of ϕ_2. The red elements mark the band width which, in this case, is 2. Elements outside this band width are not reinitialised.

Since the values for $\phi_0 < 0$ will be re-initialised too small and values $\phi_0 > 0$ will be re-initialised with values that are too big, we have a loss of mass. Thus, after each re-initialisation step we use a level set correction step.

It can be shown that the loss of mass can be compensated [11]. Let Ω_+ be that part of the domain Ω where $\Omega_+ = \{x \in \Omega : \phi_{\mathrm{r}} > 0\}$ and ϕ_{r} is the re-initialised level set. Then the level set correction C_ϕ can be computed in the following manner

$$C_\phi = \frac{S_e - S(\Omega_+)}{L(\Gamma)}.$$

S_e is the exact volume in $\mathbb{R}^3$ or area in $\mathbb{R}^2$ which is known to us. $S(\Omega_+)$ is the volume or area after the re-initialisation and $L(\Gamma)$ is the area in $\mathbb{R}^3$ or the length in $\mathbb{R}^2$ of the interface. If $S_e > S(\Omega_+)$ then $C_\phi < 0$ and the level set ϕ_{r} is to be lowered and vice versa.

6.3.6.4 Regularisation

Some of the terms in coupled Navier-Stokes - Level-Set equations are problematic from a numerical point of view. We have jump conditions for μ and ρ and need to discretise δ_Γ.

To discretise the delta distribution is problematic. We use a smoothed and weighted derivative of a Heaviside[16] function. The smoothed Heaviside function

[16]Oliver Heaviside (1850-1925).

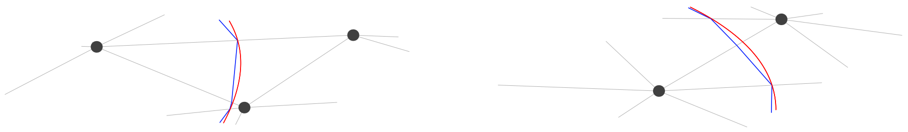


Figure 6.8: Two types of errors that may be introduced during re-initialisation. The red line represents the true ϕ_0. In blue are the segments that are found by the contour plotting utility. The segments intersect the mesh elements and are computed by a linear interpolation between nodal values. Left: Even if the intersection points are found with sufficient precision, the blue segment does not capture the true ϕ_0 due to the segment's linear nature. Right: In this case the intersection is not found correctly.
In both cases, by using the blue ϕ_0 representation an error in recomputing the distance function is made.

has the form

$$\hat{H}(x) = \begin{cases} -1 \le x \le 1; & \frac{1}{2} + x\left(\frac{15}{16} - x^2\left(\frac{5}{8} - \frac{3}{16}x^2\right)\right) \\ x \le -1; & 0 \\ 1 \le x; & 1 \end{cases}$$

and its derivative

$$\hat{H}'(x) = \begin{cases} -1 \le x \le 1; & \frac{15}{16} - x^2\left(\frac{5}{8} - \frac{3}{16}x^2\right) + x\left(\frac{3}{8}x - 2x\left(\frac{3}{8} - \frac{3}{16}x^2\right)\right) \\ \text{else;} & 0 \end{cases} .$$

To model δ_Γ we use

$$\hat{\delta}_\Gamma(x) = \frac{1}{h_w}\hat{H}'\left(\frac{x}{h_w}\right),$$

where x is the value of the level set function ϕ and h_w is the interface width and typically in the size of a few typical mesh elements. The process is depicted in Figure 6.10.

6.4 Initial & Boundary Conditions

Because of our inability to solve fully coupled non-linear equations analytically, we reverted to discrete solution techniques. This, however, also enforces us to deal with initial and boundary conditions.

Initial conditions represent the system's starting state. From this point in time the system is advanced to future states. Initial conditions are only of interest if the system of equations is time dependent. Boundary conditions represent the interaction of the system with its surroundings. Since only a finite extension of a system can be modelled, boundary conditions play the role of the interface

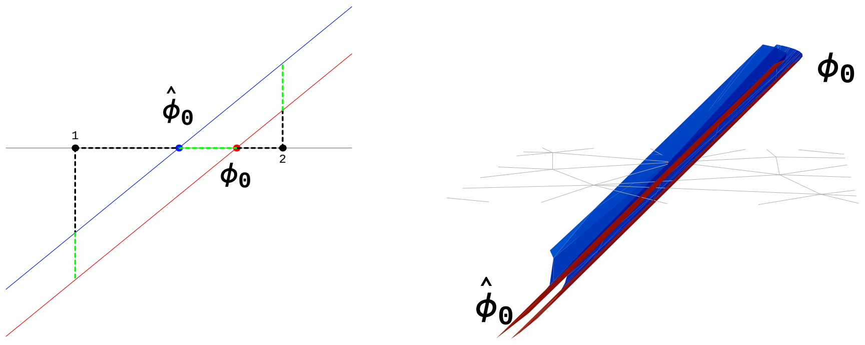


Figure 6.9: Level set loss of mass. Left: We assume the true level set ϕ_0 at the red point of intersection with the grey mesh line. Due to re-initialisation errors the assumed point of intersection is the blue point. While re-initialising the distance from mesh node 1 to the blue intersection is computed; here an error $e_1 = -|\phi_0 - \hat{\phi}_o|$ is introduced and $\hat{\phi}_o$ at mesh node 1 that is too small by e_1. Likewise at mesh node 2 the error is $e_2 = +|\phi_0 - \hat{\phi}_o|$. Consequently $\hat{\phi}_o$ at node 2 is re-initialised too high. Right: Displays the same problem in an exaggerated manner on a mesh in $\mathbb{R}^2$.

between the system and its surroundings. The surroundings are captured in the boundary conditions. Boundary conditions must always be given.

6.4.1 Boundary conditions

6.4.1.1 Dirichlet - essential conditions

To specify a value of the unknown u at $\partial\Omega$

$$u_{\partial\Omega} = f(\mathbf{x})$$

is a Dirichlet boundary condition, where $f(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$. Dirichlet boundary conditions are essential in the sense that not specifying them on some part of $\partial\Omega$ implies Neumann zero - natural - boundary conditions, which would leave the system floating. A system is called floating if only derivatives are specified at the boundary $\partial\Omega$.

6.4.1.2 Neumann - natural conditions

To specify the derivative of the unknown u at $\partial\Omega$

$$\frac{\partial}{\partial \mathbf{n}} u = f(\mathbf{x})$$

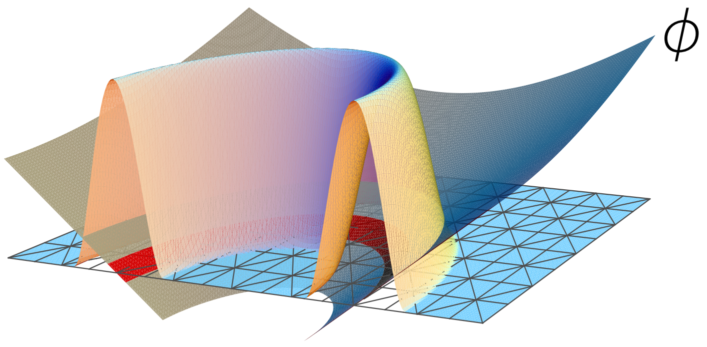


Figure 6.10: With the aid of a smoothed Heaviside function derivative the δ_Γ distribution for the surface tension term computation is modelled. The level set $\phi = \sqrt{(x-1/2)^2 + (y-1/2)^2} - 1/4$ cuts the $\mathbb{R}^2$ plane at ϕ_0, which is marked in red. The ϕ values are input to the smoothed Heaviside function. At ϕ_0 the Heaviside function has its maximum. For demonstration purposes the derivative of the Heaviside function is scaled.

is a Neumann boundary condition, where $f(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$. The special case where no boundary conditions are specified on some part of $\partial\Omega$ implies

$$\frac{\partial}{\partial \mathbf{n}} u = 0$$

This can be understood from the boundary part of Equation 6.6

$$\int_{\partial\Omega} \left((\sigma \nabla u)^T \cdot \mathbf{n} \right) w_i d\partial\Omega.$$

Ignoring the term implies the assumption that

$$\frac{\partial}{\partial \mathbf{n}} u = 0.$$

Not specifying anything thus implies a natural boundary condition, namely Neumann zero.

Besides specifying the flux over $\partial\Omega$ Neumann boundary conditions can be used to model symmetry. If a line of symmetry is present, no flux passes over this line.

6.4.1.3 Cauchy - mixed conditions

A generalisation of Dirichlet and Neumann boundary conditions is the Cauchy condition

$$\alpha \frac{\partial}{\partial \mathbf{n}} u + \beta u = f(\mathbf{x}).$$

6.4.1.4 Time dependent boundary conditions

Boundary conditions can be time dependent. This is especially true for inflow or inlet boundary conditions in the fluid flow area. It is important to evaluate the boundary conditions at the “right point in time”. Let us say we are at simulation time t_n. For an explicit time integration method we evaluate the boundary conditions at time t_n. For an implicit time integration algorithm we evaluate the boundary conditions at time $t_n + \triangle t$ where $\triangle t$ is the step size to the next time step t_{n+1}.

6.4.2 Initial conditions

So far we have dealt with spatial conditions only. Initial conditions deal with the system’s state at some point in time. Depending on which of the temporal derivatives are present, first and/or second derivative, we speak of initial conditions of the first or second kind.

6.4.2.1 Initial conditions of the first kind

Here the systems state u_0 at some point in time t_0 is prescribed

$$\tau_1 \frac{\partial}{\partial t} u \Rightarrow u_0 \forall \Omega,$$

where u_0 is the systems state a t_0 in the entire domain Ω.

6.4.2.2 Initial Condition of the Second Kind

Should a second time derivative be present - even if no first time derivative is present - we need to specify an initial velocity $\dot{u}_0$ in addition to the initial state u_0

$$\tau_2 \frac{\partial}{\partial t^2} u + \tau_1 \frac{\partial}{\partial t} u \Rightarrow \dot{u}_0,\, u_0 \forall \Omega$$

6.4.3 Fluid boundary conditions

In the fluid literature several concepts of technical initial and boundary conditions have been introduced. We quickly summarise them and show how they can be expressed using mathematical terminology.

6.4.3.1 Inlet conditions

At inlets either the pressure or the velocity components are prescribed by some fixed value. Inflow conditions may be time-dependent. We thus enforce the appropriate Dirichlet boundary conditions at each time step. Should the entire boundary $\partial\Omega$ consist of prescribed velocity conditions, it is useful to set at least one point on the boundary to a Dirichlet pressure boundary condition since else the pressure would be implicitly set to Neumann zero which would imply that the system is floating.

6.4.3.2 Outlet conditions

Employ the same mechanism as with the inlet conditions.

6.4.3.3 Walls - No-Slip

On walls we have no-slip conditions. No-Slip conditions assume that both the tangential and normal velocity component are zero. This can be achieved by setting all velocity parts to Dirichlet zero. For example $\mathbf{u} = 0$.

6.4.3.4 Walls - Free-Slip and Symmetry

In the slip (also known as symmetry) boundary condition only the normal velocity is set to Dirichlet zero as in $\mathbf{u} \cdot \mathbf{n} = 0$ and the tangential component is set (implicitly) to Neumann zero.

6.4.3.5 Initial conditions

In the case of the Navier-Stokes equations first time derivatives are present. This implies that the initial state u_0 of the system of equations must be known. It is best to start the system from an initial state at rest. Then an initial state $u_0 = 0$ in Ω is taken.

References

[1] Finlayson and Scriven. The method of weighted residuals - a review. *Applied Mechanics Reviews*, 19(9):735 – 748, 1966.

[2] Peter Hunter and Andrew Pullan. *FEM/BEM Notes*. University Of Auckland, 2001.

[3] Jianming Jin. *The finite element method in electromagnetics*. New York : Wiley, 1993.

[4] Victor N Kaliakin. *Introduction To Approximate Solutions Techniques, Numerical Modeling, and Finite Element Methods.* Marcel Dekker, 2002.

[5] Rainald Loehner. *Applied CFD Techniques: An Introduction Based on Finite Element Methods.* John Wiley & Sons, 2001.

[6] D. R. J. Owen and E. Hinton. *A simple guide to finite elements.* Swansea, UK : Pineridge, 1980.

[7] Frank Rieg and Reinhard Hackenschmidt. *Finite Elemente Analyse für Ingenieure.* Wien: Hanser, 2000.

[8] H. R. Schwarz. *Methode der finiten Elemente.* Teubner, Stuttgart, 1984.

[9] Ir. A. Segal. *Finite element methods for the incompressible Navier-Stokes equations.* J.M. Burgerscentrum, Delft University of Technology, 2004.

[10] J.A. Sethian. *Levelset methods and fast marching methods.* Cambridge University Press, 1999.

[11] Anton Smolianski. Finite-element/level-set/opereator-splitting (felsos) approach for computing two-fluid unsteady flows with free moving interfaces. *International Journal for Numerical Methods in Fluids*, Volume 48, Issue 3:231 – 269, 2004.

[12] Friedrich Weller. *Numerische Mathematik für Ingenieure und Naturwissenschaftler.* Braunschweig; Wiesbaden: Vieweg, 1996.

[13] O. C. Zienkiewicz. *The Finite Element Method.* Butterworth-Heinemann, 2000.

7 Implementation

I've learnt so much from my mistakes...
I'm thinking of making a few more
(Unknown)

7.1 Design Philosophy **92**
7.2 Program Flow **94**
7.3 Graphs **95**
7.4 Operators & Matrix Assembly **101**
7.5 Boundary Conditions **109**
7.6 Systems **112**
7.7 System Analysis **113**

IMPLEMENTATION is challenging. In the following chapter we show some aspects of the implementation. Particular attention has been paid to choosing code segments which are of central importance to understanding the concept behind the implementation. Some other code segments have been chosen since some novel ideas have been tried out.

The implementation was done in *Mathematica* and resulted in the open source IMTEK *Mathematica* Supplement, or IMS for short. The Add-On packages are available for free download [14]. Furthermore, a mailing list for further assistance is available.

We try to present the ideas in a way which is a natural work flow. In the first part the work is presented in an overview. Then we weave our way along and point out the sights.

Some portions of the code rely on the reader to have a good understanding of the *Mathematica* programming language. Some good references include Mäder [10, 11] and the tutorials in the IMS [14].

7.1 Design Philosophy

7.1.1 Less is more

In order to implement a programming project like the one presented in this work it is essential to have a direction to avoid cluttering of the program.

> Perfection is reached not when there is no longer anything to add
> but when there is no longer anything to take away.
>
> (Antoine de Saint-Exupéry)

Antoine de Saint-Exupéry's sentence is a design philosophy and thus a compass and supposed to point to the right direction. Also, the future development and maintenance of the software project must be manageable by as few people as possible which is another call for simplicity.

7.1.2 Programming style

Traditionally most numerical implementations have been done in a imperative programming language like FORTRAN or C. Imperative languages are based on Turing[1] machines.

Functional programming languages are based on Church's[2] λ-calculus. Church and Turing then proved that both the Turing machine and the λ-calculus are

[1]Alan Mathison Turing (1912 - 1954) provided an influential formalisation of the concept of algorithm and computation with the Turing machine, formulating the now widely accepted "Turing" version of the Church-Turing thesis, namely that any practical computing model has either the equivalent or a subset of the capabilities of a Turing machine.

[2]Alonzo Church (1903 - 1995). The lambda calculus emerged in his famous 1936 paper showing the existence of an "undecidable problem". This result preempted Alan Turing's famous work on the halting problem which also demonstrated the existence of a problem unsolvable by mechanical means. He and Turing then showed that the lambda calculus and the Turing machine used in Turing's halting problem were equivalent in capabilities, and subsequently demonstrated a variety of alternative "mechanical processes for computation." This resulted in the Church-Turing thesis.

equivalent. Nevertheless, the ability to generate and modify functions is unique to functional languages like Lisp. This idea leads eventually to the vanishing of the barriers between data structure and function. The relation between functional and imperative programming is summed up [1]

> Pascal is for building pyramids - imposing, breathtaking, static structures built by armies pushing heavy blocks into place. Lisp is for building organisms – imposing, breathtaking, dynamic structures built by squads fitting fluctuating myriads of simpler organisms into place. The organizing principles used are the same in both cases, except for one extraordinarily important difference: The discretionary exportable functionality entrusted to the individual Lisp programmer is more than an order of magnitude greater than that to be found within Pascal enterprises. Lisp programs inflate libraries with functions whose utility transcends the application that produced them. [...] As a result the pyramid must stand unchanged for a millennium; the organism must evolve or perish.
> (Alan J. Perlis)

The choice in favour of the programming environment *Mathematica* was made since it is an advanced programming language in the sense that it does not try to restrict the programmer to one programming paradigm but offers a wealth of programming techniques from imperative over functional or rule based programming to pattern matching and logical paradigms. *Mathematica* being fully garbage collected relieves the programmer of another burden.

The disadvantage is that the *Mathematica* programming language is closed source and thus no bugs can be fixed. Also it is not predictable what function names future versions will provide and thus it was decided to prefix every newly added function with `ims` which stands for IMTEK *Mathematica* Supplement. From a temporal and spatial performance perspective *Mathematica* is not able to match specialised C or even Assembler code. However, using vectorised computations *Mathematica* is faster than commonly anticipated.

However, programming language development advances and one promising alternative in the future will be the Lisp Universal Shell (LUSH), which offers the capability to automatically generate C code from Lisp and interface existing C libraries.

7.1.3 Abstract programming via constructors & selectors

Throughout the implementation the constructor/selector paradigm has been used. Each new data type introduced is created with a constructor. To select from the newly generated data type selectors are employed. With this simple mechanism it is possible to change the underling data structure without modifying the parts of the code that are built on top of it. It is also evident from this that a careful design of the interface - the constructors, selectors and mutators - has to be made [1]. All constructors are characterised by a `Make` in their function name, all selectors show a `Get` in their name. Mutators, which may re-set some values of a data structure have a `Set` in their name.

Function names that are *Listable*[3] have the postfix s attached to their names. So the function `imsGetNodeIds` operates on either one node or a list of nodes returning one node's id or a list of node ids, respectively. This, in retrospect, was a poor design choice; wrongly spelt words like `imsGetDatas` contribute more to confusion than anything else.

The constructor/selector paradigm has, at least in Mathematica, the disadvantage that it comes at some cost of performance. So that this cost is not usury we compromised on not necessarily using the constructor/selector paradigm if we are inside one package. From the outside, however, it is strictly used.

7.2 Program Flow

The core program and work flow can be summed up in the following steps which we call the design cycle:

1. Create a Graph from nodes and elements:
   a) Manual creation:
      `imsMakeNexus[boundaryNodes,interiorNodes,elements]` $\rightarrow$ `imsNexus`
   b) Automatic creation based on geometry data:
      `imsGraphics3DToNexus[Graphics3D]` $\rightarrow$ `imsNexus`
2. Apply operators $\mathcal{L}$ to graph elements i:
   `imsMakeElementMatrix[`$\mathcal{L}$`[imsNexus,`i`]]` $\rightarrow$ `imsElementMatrix`
3. Assemble local element matrices into empty global matrices A_e:
   `imsAssemble[`$\{$`imsElementMatrix,...`$\}, A_e$`]` $\rightarrow A$
4. Create systems of equations:
   `imsMakeSystem[`f, S, D, M`]` $\rightarrow$ `imsSystem`
5. Analyse system of equations:
   a) Stationary analysis:
      `imsStationarySolve[imsSystem]` $\rightarrow$ `List`

[3]A function has attribute *Listable* if it can be threaded - that is applied - over a list of its input arguments.

b) Transient analysis:
   `imsTimeIntegrate[imsSystem]→ List`
c) Harmonic analysis:
   `imsHarmonicSolve[imsSystem]→ List`
d) Eigenwert analysis:
   `Eigensystem[`A`]→ List`

6. Visualise

   a) 2D:
      `List→ Graphics`
   b) 3D:
      `List→ Graphics3D`

In the next sections we fill this skeleton and show how the steps interface with each other and with themselves.

7.3 Graphs

Graphs consist, as elaborated in Chapter 2.1 on page 12, of nodes and elements. Also, in the implementation we call a graph a nexus[4]. In the following we first introduce nodes and then different element libraries. We then introduce the mechanism of how to implement new element libraries.

7.3.1 Nodes

The implementation of a node is realised by means of a constructor.

```
imsMakeNode[id_,coords_,marker_Integer:0,value_:{{0.}},data___] /;
   MatrixQ[value] := imsNode[id,coords,marker,value,data];
```

Each node has its identification (id) and its coordinates (coords) as mandatory input parameters. Both have no type requirement but commonly an integer and a numerical list, respectively, are chosen. Furthermore, an integer as a marker which defaults to 0 and a value which defaults to a 1×1 matrix with value numerical $0.$ can be given. The marker may, for example, be used to specify to which part of a boundary a node belongs. Values are to store the result of a

[4]Etymologically nexus comes from Latin and means connection, link. The fact that a graph is called nexus has historical reasons. First the implementation was called graph which conflicted with provided *Mathematica* functions. The data structure was then renamed nexus. It was only after this conversion that all ims functions where prefixed with `ims` to generate unique function names; renaming the nexus to imsGraph was then a too delicate code refactoring since virtually everything is based on the nexus data structure.

simulation. The creation of the node will fail if the given value is not of matrix form. As a last input specification data of arbitrary type can be given. The node data type is returned. For each input parameter a selector exists and for some a mutators exists This will be illustrated.

Loading the node package	`In[1]:= Needs["Imtek‘Nodes‘"]`

Creation of a node with identification number 1 and the coordinates $\{0, 0\}$.	`In[2]:= n = imsMakeNode[1,{0,0}]`
	`Out[2]= imsNode[1,{0,0},0,{{0}}]`

Selection of the node's id.	`In[3]:= imsGetIds[n]`
	`Out[3]= 1`

A node may hold more than one value. The additional degrees of freedom are stored in the columns of the node's value. If the node should hold transient simulation results those are stored in the rows of the node's values.

Resources:	
Implementation	Nodes.nb\.m
Documentation	NodesDocu.nb
Examples	see Nexus

7.3.2 Element library types

7.3.2.1 Mesh element library

The main purpose of the mesh element library is to provide a data structure for partial differential equation operators built on mesh based discretisation methods as discussed in Chapter 6. The mesh elements and the corresponding nodes will be input to partial differential equation operators.

As an example we inspect the 1 degree of freedom per node triangle element. The principles presented hold for all other mesh elements. The constructor is defined as

```
imsMakeTriangleLinear1DOF[id_Integer,nodesIds_List,marker_Integer:0,datas___] :=
   imsTriangleLinear1DOF[id,nodesIds,marker,datas];
```

This creates a linear triangle mesh element with 1 degree of freedom nodes. Each element has a unique identification number (id) and a list of node ids which form the element's vertices. Markers can be used to specify the affiliation of an element to a specific area in the simulation domain (see convection diffusion example in Chapter 4). The marker defaults to integer 0. Optional data can be given. A triangle mesh element is returned.

Remark: Strictly speaking keeping the degrees of freedom as part of the mesh element name is cumbersome and not necessary since the degrees of freedom are transported by the nodes alone; and they can carry the burden.

One further issue needs to be addressed, namely the visualisation of nodes. To tie in with constructors and selectors we call them representors. The code for the triangle element is

```
imsDrawElements[te_imsTriangleLinear1DOF,nodes_] :=
   Line[imsClosePolygon[imsGetCoords[nodes]]];
```

`imsGetCoords` returns the coordinates of the nodes. To close the coordinates to a closed line segment the function `imsClosePolygon` is engaged.

Resources:	
Implementation	MeshElementLibrary.nb\.m
Documentation	MeshElementLibraryDocu.nb

7.3.2.2 Domain element library

The purpose of domain elements is to unify the input data structure for mesh generation. The simulation domain's boundary $\partial\Omega$ is described by domain segments.

To generate the necessary input in 2 dimensions the function `imsConvex-Intersect` is available and is further subsidised by the hosting package `Imtek`-Polygon`. It is thus possible to intersect convex polygons in $O(n \log n)$ temporal complexity. In $3$ dimensions the routine `imsGraphics3DToNexus` converts arbitrary 3 dimensional graphics objects into a graph made up from domain segments. We can thus draw on the wealth of mathematical functions provided by *Mathematica* itself or extensions such as shown by Barrere [3]. The conversion is performed by a unique hash-table[5] based node creation which is elucidated in the following example

We create a function f with coordinates as arguments and assign an integer.	`In[1]:= f[{1,0}] = 1;`
	`In[2]:= f[{2,0}] = 2;`

Each time a function is called it evaluates to the assigned integer.	`In[3]:= f[{2,0}]`
	`Out[3]= 2`

This process is utilised in the function `imsPolygonToDomainSegments` which converts a polygon given by coordinates to a list of boundary nodes and domain segment elements.

[5]Hash-tables have a temporal complexity of $O(1)$ and are thus very fast data structures. This then allows for a fast conversion of `Graphics3D` objects to input for mesh generators.

```
imsPolygonToDomainSegements[polygons_] := Module[
   {i,f,boundaryNodes,domainSegments,coords,incidents},
   coords = Union[Flatten[List @@@ Flatten[polygons],2]];
   i=1;
   (f[#]=i++)& /@ coords;
   incidents = Map[f,(Sequence @@@ polygons),{2}];
   i=1;
   boundaryNodes = imsMakeNode[i++,#]& /@ coords;
   i=1;
   domainSegments = imsMakeDomainSegment[i++,#]& /@ incidents;
   Return[{boundaryNodes,domainSegments}];];
```

We find the sorted union of all coordinates which is a process of $O(n \log n)$ complexity and after which duplicate coordinates have been eliminated. Then we create functions named f which, when called with any one of the coordinates as arguments, return the unique integer id i. f is then mapped to the sequence of the polygon at level 2 which then returns the incidents. We can proceed by creating the boundary nodes and the domain segments and return a nexus.

Unfortunately, *Mathematica* itself does not provide selectors for `Graphics3D` objects. So, brute force is the only resort. We convert the `Grapichs3D` to a `List` and extract the polygons which are then given to `imsPolygonToDomainSegments`.

The conversion of a `Graphics3D` object is then done by

```
imsGraphics3DToNexus[gr_Graphics3D] := Module[
   {polygons,boundaryNodes,elements},
   polygons = Cases[Flatten[List @@gr],_Polygon];
   {boundaryNodes,elements} = imsPolygonToDomainSegements[polygons];
   Return[imsMakeNexus[boundaryNodes,{},elements]];];
```

In the following code segments we demonstrate the procedure

We import a stl file.

```
In[1]:= g = Import["Giraffe.stl"];
```

We load the appropriate packages and convert the `Graphics3D` to an `imsNexus`

```
In[2]:= Needs["Imtek`DomainElementLibrary`"];
In[3]:= Needs[
        "Imtek`Interfaces`TetgenInterface`"];
```

We convert the `Graphics3D` to an `imsNexus`

```
In[4]:= domainSegementNexus =
        imsGraphics3DToNexus[g];
```

The nexus is converted to the mesh generator tetgen's input file, exported and run through tetgen.

```
In[5]:= meshFile = imsToTetgenInputFile[
        domainSegementNexus];
In[6]:= filePath = Export["test.poly",
        meshFile,"Table"];
In[7]:= Run["tetgen -pq "<>filePath];
```

The mesh is loaded back and we query the number of elements

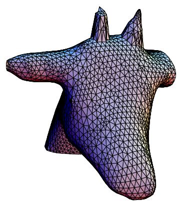
```
In[8]:= tetgenMesh = imsReadTetgenOutput[
        {"test.1.node","test.1.ele"}];
In[9]:= Length[imsGetElements[tetgenMesh]]
Out[9]= 730937
```

To depict the surface mesh we load the faces file and the coordinates and remove unneeded data.

```
In[10]:= faces = Drop[Cases[Import[
         "test.1.face","Table"],
         {_Integer,___}],1][[All,{2,3,4}]];
In[11]:= coords = Cases[Import[
         "test.1.node","Table"],
         {_,_,_,_,_}][[All,{2,3,4}]];
```

We display the new surface mesh.

```
In[12]:= Show[Graphics3D[Polygon/@
         (coords[[#]]&/@faces)]];
```



Remark: The stl files need to be well-behaved in the sense that no intersecting polygons are allowed.

Resources:	
Implementation	DomainElementLibrary.nb\.m
Documentation	DomainElementLibraryDocu.nb

7.3.2.3 Circuit element library

In principle the circuit element library behaves in the same manner. The difference here is the need of circuit elements to carry a value in which, for example, the resistor carries its resistance. The constructor is then

```
imsMakeResistance[id_Integer,nodesIds_List,values_,datas___] :=
   imsResistance[id,nodesIds,values,datas];
```

Resources:	
Implementation	CircuitElementLibrary.nb\.m
Documentation	CircuitElementLibraryDocu.nb

7.3.2.4 Creating new element libraries

To extend the libraries the reader must know the following rules.

- Each element needs an identification number and a list of incidents specifying the identification numbers of the nodes spanning the element.

- For both id and incidents at least the selectors `imsGetIds` and `imsGet-IncidentsIds` must be defined
- The function `imsDrawElements` must be defined

Consider the following definition of a new element

```
MakeMyNewElement[id_,nodes_]:=MyNewElement[id,nodes];
imsGetIds[e_MyNewElement]:=e[[1]];
imsGetIncidentsIds[e_MyNewElement]:=e[[2]];
imsDrawElements[e_MyNewElement,nodes_]:=myDraw[imsGetCoords[nodes]];
```

The myDraw routine is problem dependent and specifies the way in which the connection of the coordinates of the element's nodes are drawn.

Resources:	
Examples	see Nexus

7.3.3 Nexus

7.3.3.1 The nexus data structure

In essence the nexus is little more than a wrapper.

```
imsMakeNexus[boundaryNodes_List,interiorNodes_List,elements_List] :=
   imsNexus[boundaryNodes,interiorNodes,
   Sort[elements,OrderedQ[imsGetIds[{#1,#2}]]&],
   Sort[Join[boundaryNodes,interiorNodes],
   OrderedQ[imsGetIds[{#1,#2}]]&]];
```

We see that the nexus contains the boundary and interior nodes, a sorted list of the elements and a duplicated but sorted list of the joined boundary and interior nodes.

It is instructive to investigate the internal functioning of the imsDrawElements procedure

```
imsNexus/:imsDrawElements[a_imsNexus,ids___] :=
   imsDrawElements[#,imsGetNodes[a,imsGetIncidentsIds[#]]]&
   /@ imsGetElements[a,ids]
```

The `imsDrawElements` procedure is overloaded[6] for a nexus expression. By these means it is further possible to specify a range of element ids of a nexus to be drawn.

[6]We speak of overloaded when two procedures are given the same name in order to operate on different levels of data structures. In this specific case the `imsNexus` is a higher level data structure than the elements, since the nexus is made up of elements.

7.3.3.2 Future extensions

For now the graph representation as presented is limited. Mainly the disability to contain sub-circuits is a lack of flexibility. Sub-circuits would be of great attraction for lumped sub-circuits. This could be overcome by implementing the ability that a nexus may contain a nexus, perhaps recursively.

Resources:	
Implementation	Graph.nb\.m
Documentation	GraphDocu.nb

7.4 Operators & Matrix Assembly

Operators transform graph elements to local element matrices. Each operator, as a bare minimum, must thus return an element matrix for further processing in the matrix assembly step. The element matrix is a mere wrapper

```
imsMakeElementMatrix[values_?MatrixQ,rows_List,columns_List] :=
  imsElementMatrix[values,rowPos,columnPos];
```

For all data of the constructor appropriate selectors exist.

Example: The destination of the local element values $\{\{e_{11}, e_{12}\}, \{e_{21}, e_{22}\}\}$ in the global matrix A are stored in the rows $\{2, 3\}$ and columns $\{3, 4\}$. This specifies that in the global matrix A the following entries must be set

$$A_{2,3} \leftarrow e_{11},\ A_{2,4} \leftarrow e_{12},\ A_{3,3} \leftarrow e_{21} \text{ and } A_{3,4} \leftarrow e_{22}.$$

The matrix assembly step is more intricate; not because it is theoretically difficult but because temporal performance is absolutely crucial and a direct, imperative implementation is thus not feasible in *Mathematica*. The following commented code shows the workings of a vectorised matrix assembler.

An empty matrix and two element matrices are created.

```
In[1]:= matrix = Table[0,{i,3},{j,3}];
In[2]:= elems = {imsMakeElementMatrix[
          {{a11,a12},{a21,a22}},{1,2},{1,2}],
          imsMakeElementMatrix[
         {{b11,b12},{b21,b22}},{3,2},{3,2}]};
```

We create a helper function that generates the matrix positions from the rows and columns the element is supposed to go to.

```
In[3]:= convertToInci[
          imsElementMatrix[ma_,r_,c_]]:=
          Outer[List,r,c];
```

The incidents are extracted and flattened to level 2. The ordering is found and stored. The ordered incidents are then split.

```
In[4]:= incidents = Flatten[convertToInci
        /@ elems,2];
In[5]:= orderedList = Ordering[incidents]
Out[5]= {1,2,3,4,8,7,6,5}
In[6]:= incidents = Split[
        incidents[[orderedList]]]
Out[6]= {{{1,1}},{{1,2}},{{2,1}},{{2,2},
        {2,2}},{{2,3}},{{3,2}},{{3,3}}}
```

The incidents are now split in a manner where duplicate entries are consecutive.

The values are extracted and ordered.

```
In[7]:= vals = Flatten[
        imsGetElementMatrixValues
        /@ elems][[orderedList]]
Out[7]= {a11,a12,a21,a22,b22,b21,b12,b11}
```

As a next step we would like to add the values which point to the same entry in the global matrix. For this we use the fact that the duplicate entries are consecutive.

Starting from 1 we fold `Plus` around the list of length of global index positions. We generate input for `Take` and apply `Plus` to the taken values.

```
In[9]:= startPos = FoldList[Plus,1,Length
        /@ incidents]
Out[9]= {1,2,3,4,6,7,8,9}
In[10]:= takePos = Drop[Transpose[
        {#,RotateLeft[#]-1}&[startPos]],
        -1]
Out[10]= {{1,1},{2,2},{3,3},{4,5},{6,6},
         {7,7},{8,8}}
In[11]:= addedVals = (Plus @@
         Take[vals,#])& /@ takePos
Out[11]= {a11,a12,a21,a22+b22,b21,b12,b11}
```

The sparse matrix is created and added to the original matrix

```
In[12]:= matrix += SparseArray[
         incidents[[All,1]]->addedVals,
         Dimensions[matrix]]
```

Resources:	
Implementation	Assembler.nb\.m
Documentation	AssemblerDocu.nb

7.4.1 Numerical integration

Not much needs to be said here: For the triangular based elements the Tables 6.3 and 6.4 are implemented and for the quad based the details given in Chapter 6.3.3 are implemented.

Resources:	
Implementation	NumericalIntegration.nb\.m
Documentation	NumericalIntegrationDocu.nb

7.4.2 Shape functions

Motivated by the fact that for the numerical integration in the finite element operators the pre-integrated shape functions are used, we present the mechanism, i.e. the function `imsShapeFunctionSymbols` to automatically generate the necessary symbols. Via these symbols we can then access, among other things, the pre-integrated shape function. As an input to `imsShapeFunctionSymbols` we give an arbitrary name, the shape function - for example one from Table 6.1, the integration points and integration weight.

First we load the package and define a shape function. The shape function is then used to generate the necessary functions which are attached to the `imsLineLinear1DOF` symbol.

```
In[1]:= Needs["Imtek`ShapeFunctions`"];
In[2]:= sf = Function[{r},{(1-r)/2,(1+r)/2}];
In[3]:= imsShapeFunctionSymbols[
        imsLineLinear1DOF,sf,
        {{1/Sqrt[3]},{-1/Sqrt[3]}},{1,1}];
```

Now, several symbols have been created. `imsShapeFunction` returns the defined shape function.

```
In[4]:= imsShapeFunction[
        imsLineLinear1DOF]
Out[4]= Function[{r},{(1-r)/2,(1+r)/2}]
```

`imsShapeFunctionDerivative` returns the derivative of the defined shape function.

```
In[4]:= imsShapeFunctionDerivative[
        imsLineLinear1DOF]
Out[4]= Function[{r},{(-1/2,1/2}]
```

`imsNIntegratedShapeFunction` returns the numerical values of the shape function evaluated at the integration points.

```
In[5]:= imsNIntegratedShapeFunction[
        imsLineLinear1DOF]
Out[5]= {{0.211325,0.788675},{0.788675,
        0.211325}}
```

Also the functions `imsIntegrationWeights` and `imsIntegratedShapeFunctionDerivative` are generated for the element. Each generated function exists in a symbolic and in a numeric version, made visible through the capital N in the function name. The automatic differentiation is handled by the following lines of private[7] code.

```
SetAttributes[sFD,Listable];
sFD[shapeFunction_] :=
   Derivative[Sequence@@#][shapeFunction]& /@
   Outer[KroneckerDelta,Range[#],Range[#]]&
   [Length[shapeFunction[[1]]]];
```

Here the derivatives are generated symbolically in all directions of space held by the shape function. Some excerpts of the code of the function `imsShapeFunctionSymbols` are presented.

[7]Private means that the function is only defined locally, in the `Private`` context and not exported to the `Global`` context.

```
imsShapeFunctionSymbols[name_Symbol,function_,quadPoints_,quadWeights_] :=
   CompoundExpression[imsShapeFunction[name] = function,
   ...
   imsShapeFunctionDerivative[name] = sFD[imsShapeFunction[name]],
   ...
   imsIntegratedShapeFunction[name] = imsShapeFunction[name][Sequence@@#]&
   /@ quadPoints,
   imsNIntegratedShapeFunction[name] = N[imsIntegratedShapeFunction[name]],
   imsIntegratedShapeFunctionDerivative[name] =
   Through[imsShapeFunctionDerivative[name][Sequence @@ #]]&
   /@ quadPoints,
   ...
   imsIntegrationWeights[name]=quadWeights,
   ...]
```

Resources:	
Implementation	ShapeFunctions.nb\.m
Documentation	ShapeFunctionsDocu.nb

7.4.3 Finite element operators

A generic skeleton for a finite element operator is presented next. First the coordinates and the markers are extracted from the nodes and the element, respectively. Next the head of the element is read. The element names are chosen to be the shape function names. This is the means by which the shape function steps into the operator. Thereafter, we extract the values as well as the row and column incidents of this local element. Now we look up the above defined pre-integrated shape functions and their derivatives which are attached to the above generated symbols. The integration weights are also loaded. Since the shape functions are defined in local coordinates we do a coordinate transformation. To this end we compute the Jacobians via Equation 6.16 and find the determinants and the inverses. The operator's input function needs to be integrated. This is done by applying the operator's input function to the mapping created with the dot product of the integrated shape function and the coordinates. We also note that here the marker is handed over to the input function, for the user to specify input functions dependent on the mapped coordinates and/or the marker of an element.

```
imsFEMxyOperator[{inESM_,inERHS_},elem_,elementNodes_,coefficient_] :=
   Block[{....},

   (* element data *)
   coords = imsGetCoords[elementNodes];
   marker = imsGetMarkers[elem];
   sfElement = Head[elem];
(* retrieve element parts *)
outESMvalues = imsGetElementMatrixValues[inESM];
rows = imsGetElementMatrixRows[inESM];
cols = imsGetElementMatrixColumns[inESM];

(* shape functions *)
sf = imsIntegratedShapeFunction[sfElement];
sfrsderiv = imsIntegratedShapeFunctionDerivative[sfElement];
integrationWeight = imsIntegrationWeights[sfElement];

(* mapping *)
jacobians = Transpose[(#.coords)]&/@sfrsderiv;
jDets = Det[#]&/@jacobians;
jInverses = Inverse[#]&/@jacobians;

(* function integration *)
coefficientVals = (coefficient @@ Flatten[{marker,#}])& /@ (sf.coords);

(* integration *)
....

Return[{imsMakeElementMatrix[outESMvalues,rows,cols],inERHS}];];
```

The actual integration is presented in the following sections. What remains is the output of the newly computed local element matrix. For the stiffness, damping, and mass matrices the `outESMvalues` are used. For a load element matrix, or rather vector, the `outERHvalues` are used[8].

Remark: In the course of this work a finite element operator returns either a local element matrix for the global stiffness, damping or mass matrix or a contribution to the load vector. The respective other returned value is empty. If we wish, in contrast to this work, to deal with non-linearities on an element level we may do so. In such a case both the local element matrix and load contribution may be different from zero. It is thus possible to alter the linearisation process and yet not leave the presented operator framework.

Now we proceed with a close look at the specific implementations for the different operators.

7.4.3.1 Diffusion

The integration for the diffusion operators is given by

[8]The reason why the two input element matrices are called `inESM` and `inERHS` is historical. If you write a differential equation in a more traditional manner such as $\frac{\partial u}{\partial t} + (\nabla\sigma)^T \cdot \nabla u = f$ then you have the load vector f on the right-hand side. Thus the name E(lement) R(ight) H(and) side, where the other acronym stands for E(lement) S(tiffness) M(atrix).

```
(* integration *)
Do[ (* deriv of sf *)
sfxyderiv = Transpose[jInverses[[step]]].sfrsderiv[[step]];
weight =  integrationWeight[[step]]*jDets[[step]];
outESMvalues += Transpose[sfxyderiv].coefficientVals[[step]].sfxyderiv*weight;,
{step,Length[integrationWeight]}];
```

7.4.3.2 Convection

The integration for the convection operators is given by

```
   (* integration *)
   Do[(* deriv of sf *)
   sfxyderiv = Transpose[ jInverses[[ step ]] ]. sfrsderiv[[ step ]];
   weight =  integrationWeight[[ step ]] * jDets[[ step ]];
   outESMvalues += Transpose[{sf[[step]]}] .
   (coefficientVals[[step]] . sfxyderiv) * weight;,
   {step,Length[integrationWeight]};];
```

7.4.3.3 Reaction & Transient

The integration for both transient and the reaction operators is given by

```
   (* integration *)
   outESMvalues += Plus @@ (coefficientVals *
   (Dot[Transpose[{#}],{#}]& /@ sf) * integrationWeight * jDets);
```

7.4.3.4 Load

The integration for the load operators is given by

```
   (* integration *)
   outERHSvalues += Plus @@ (coefficientVals*sf*integrationWeight*jDets);
   Return[{inESM,imsMakeElementMatrix[outERHSvalues,rows,{1}]}];
```

7.4.4 Lumped operators & creation of lumped systems

In the case of lumped elements only one operator is really necessary. The operator will be overloaded for each lumped element. In essence the lumped elements store the stamps presented in Chapter 6.1. For lumped systems the implementational complexity lies not so much in the operators as it does in the creation of the system; quite in contrast to the partial differential equation operators. As an example, the implementation for a resistor is shown.

```
imsLumpedOperator[r_imsResistance] :=
   imsMakeElementMatrix[1/imsGetValues[r]*{{1,-1},{-1,1}},
   imsGetIncidentsIds[r],imsGetIncidentsIds[r]];
```

The following code segments show most parts of the lumped system creation. First we filter the options to see what kind of analysis is required and what the frequency symbol for the harmonic analysis is. It is important to distinguish between the different analysis methods since this is the mechanism which creates systems suitable for different kinds of analyses. For brevity's sake the variable definitions have been deleted.

```
imsMakeLumpedSystem[nexus_imsNexus,opts___] :=
   Module[{...},
   (* options definitions *)
   {analysisMethod,harmonicSymbol} = {imsAnalysisMethod,imsHarmonicSymbol}
    /. Flatten[{opts}] /. Options[imsMakeLumpedSystem];
   If[harmonicSymbol === Automatic,
      fVariable = I * Global`\[Omega],
      fVariable = harmonicSymbol];
```

From the length of the nodes we get the degrees of freedom and thus the initial size of the system matrices. We select from the nexus those elements which are designated for the damping matrix and assemble them into the global damping matrix. Even for a stationary analysis we start from the damping matrix; the matrix will then later be copied into the appropriate place.

```
   (* element extraction *)
   nexusLength = Length[imsGetNodes[nexus]];
   nexusElements = imsGetElements[nexus];

   (* damping matrix - resistances *)
   dampingMatrix = Table[0,{nexusLength},{nexusLength}];
   elemsDamping = imsLumpedOperator[Select[nexusElements,imsLumpedDampingQ]];
   imsAssemble[elemsDamping,dampingMatrix];
```

In the next step we select the load elements, assemble them and enlarge the system of equations if there are voltage sources. Also the load is adjusted to the voltage value.

```
    (* load vector - current sources *)
   load = Table[0,{nexusLength},{1}];
   elemsLoad = imsLumpedOperator[Select[nexusElements,imsLumpedLoadQ]];
   imsAssemble[elemsLoad,load];
   loadFlatten = Flatten[load];

   (* voltage sources *)
   imsLagrangeMultipliers[{dampingMatrix,loadFlatten},
      {imsGetIncidentsIds[#]},{imsGetValues[#]},{1,-1}]& /@
      Select[nexusElements,imsLumpedPotentialQ];
```

Next we select the voltage controlled voltage sources. If there are any present the system is first enlarged and then the appropriate values are built into the matrices. This procedure is applied in a similar manner to the current controlled current source and the current controlled voltage source.

```
 (* voltage controlled voltage source *)
elemsVCVS = Select[nexusElements,imsLumpedVCVSQ];
If[Length[elemsVCVS] > 0,
   imsLagrangeMultipliers[{dampingMatrix,loadFlatten},
      {imsGetIncidentsIds[#]},{0},{0,0,0,0} ];
   lengthDamping = Length[dampingMatrix];
   dampingMatrix[[lengthDamping,imsGetIncidentsIds[#][[1]]]] =
      -imsGetValues[#];
   dampingMatrix[[lengthDamping,imsGetIncidentsIds[#][[2]]]] =
      imsGetValues[#];
   dampingMatrix[[lengthDamping,imsGetIncidentsIds[#][[3]]]] = 1;
   dampingMatrix[[lengthDamping,imsGetIncidentsIds[#][[4]]]] = -1;
   dampingMatrix[[imsGetIncidentsIds[#][[3]],lengthDamping]] = 1;
   dampingMatrix[[imsGetIncidentsIds[#][[4]],lengthDamping]] = -1;,
   Null ] &/@ elemsVCVS;

(* current controlled current source *)
...
(* current controlled voltage source *)
...
```

Wires enlarge the system and, again, this is done via the Lagrangian multipliers.

```
 (* wires *)
imsLagrangeMultipliers[{dampingMatrix,loadFlatten},
   {imsGetIncidentsIds[#]},{0},{{1,-1}}]& /@
   Select[nexusElements,imsLumpedConnectorQ];
```

If the analysis method is stationary the stiffness elements of the lumped circuit are added by enlarging the system further. This is due to the fact that in a stationary analysis inductances are replaced by wires; setting the inductance to 0 is not an option since $1/L$ would result in a division by zero.

```
 (* lumped matrix elements short circuited for stationary solution *)
If[analysisMethod === "imsStationary",
   imsLagrangeMultipliers[{dampingMatrix,loadFlatten},
     {#},{0},{{1,-1}}]& /@ Partition[Flatten[imsGetIncidentsIds[
     Select[nexusElements,imsLumpedStiffnessQ]]],2];,
   Null;];
```

Now that all “enlarging” elements have been built into the system of equations, we can insert ground as a Dirichlet boundary condition[9]. We now know the final size of the system of equations.

[9]The method in which the Dirichlet boundary condition is inserted is not complete. If a Dirichlet condition is to be inserted then actually row and column i of the mass and damping matrix need to be set to 0, which is neglected in the current code fragment.

```
  (* ground *)
imsDirichlet[{dampingMatrix,loadFlatten},
  imsGetIncidentsIds[#],{imsGetValues[#]},1]& /@
  Select[nexusElements,imsLumpedGroundQ];
lengthDamping = Length[dampingMatrix];
load = Partition[loadFlatten,1];
```

If the analysis method is either transient or harmonic we need to assemble the inertia/mass matrix and the stiffness matrix and assemble them into their respective global matrices. This can only be done now, since it is only after including all the "enlarging" elements that we know the final size of the system.

```
  (* create inertia and stiffness matrix *)
Which[analysisMethod === "imsTransient" ||
      analysisMethod === "imsHarmonic",

   (* inertia matrix - capacitances *)
   inertiaMatrix = Table[0,{lengthDamping},{lengthDamping}];
   elemsInertia = imsLumpedOperator[Select[nexusElements,
      imsLumpedInertiaQ]];
   imsAssemble[elemsInertia,inertiaMatrix];

   (* stiffness matrix - inductances *)
   stiffnessMatrix = Table[0,{lengthDamping},{lengthDamping}];
   elemsStiffness = imsLumpedOperator[Select[nexusElements,
      imsLumpedStiffnessQ]];
   imsAssemble[elemsStiffness,stiffnessMatrix];];
```

In the last step we create the system according to the analysis method. It is noteworthy that for the stationary analysis we insert the damping matrix into the place of the stiffness matrix and thus solve directly for e and not λ.

```
  (* return condition *)
Which[analysisMethod === "imsTransient",
   Return[imsMakeSystem[load,
      stiffnessMatrix,dampingMatrix,inertiaMatrix]];,
   analysisMethod === "imsHarmonic",
   Return[imsMakeSystem[load,
    1/(fVariable) * stiffnessMatrix + dampingMatrix + (fVariable) *
    inertiaMatrix]];,
  analysisMethod === "imsStationary",
  Return[imsMakeSystem[load,dampingMatrix]];,
  True,
  Message[imsMakeLumpedSystem::"analysisMethod",analysisMethod];];];
```

7.5 Boundary Conditions

We consider two kinds of boundary conditions.

7.5.1 Dirichlet

To implement Dirichlet boundary conditions three possibilities exist of which two have been implemented. The first method is called the engineers method [8]. In practise this is nowadays seldom used and thus not further discussed.

The second method is to rearrange the equations. This method will always enforce the exact Dirichlet boundary conditions prescribed - even if they are unphysical[10]. The third method, the Lagrangian multiplier method, will allow for softer Dirichlet boundary conditions in the sense that unphysical settings will be smoothed out.

7.5.1.1 Equation rearrangement

The function `imsDirichlet` implements equation rearrangement. Assume we have a system of equations

$$A \cdot u = f = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}.$$

Now, we would like the Dirichlet boundary condition $u_2 = 3$. We thus proceed in the following manner

$$\begin{pmatrix} a_{11} & 0 \\ a_{21} & 0 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 - a_{12}u_2 \\ f_2 - a_{22}u_2 \end{pmatrix}.$$

Since u_2 is specified we have two possibilities. We could now delete row and column 2 of matrix A and row 2 of load vector f and then solve for the remaining unknowns, in this case u_1. The advantage is that the system of equations is smaller. In contrast this would leave us with reinserting the known Dirichlet value u_2 into the solution. If one wants to avoid the reinsertion it it possible to proceed as follows

$$\begin{pmatrix} a_{11} & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 - a_{12}u_2 \\ 3 \end{pmatrix},$$

which leaves the order of the solution vector unaffected. The core of the vectorised implementation looks as follows

```
matrix[[All,pos]] = 0;
matrix[[pos,All]] = 0;
matrix += SparseArray[Transpose[{pos,pos}]->
Table[diagVal,{Length[pos]}],Dimensions[matrix],0];
```

[10]Unphysical means that between two Dirichlet specified nodes the values must be sufficiently smooth.

The fact that we use a `SparseArray` to construct the diagonal values is purely for performance reasons. The variable `diagVal` has a default value of numeric 1. The default numeric 1 was chosen for performance reasons. If the user needs purely symbolic computations a value of integer 1 can be seamlessly given.

7.5.1.2 Lagrangian multipliers

Another option is to introduce additional degrees of freedom. If we take the above example

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$

where u_2 is subject to the constraint $u_2 = 3$. This can be written in the form

$$B \cdot \mathbf{x} = \mathbf{h}$$

where B is the boundary condition matrix. The expanded form is

$$\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = (3)\,.$$

We introduce a set of quantities λ_i, called Lagrangian multipliers, by

$$\lambda^T \left(B\mathbf{x} - \mathbf{h} \right) = 0.$$

This can be considered as the energy required to maintain the boundary conditions. We now assume we have

$$I = \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{f},$$

adding both

$$J = \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{f} + \lambda^T \left(B\mathbf{x} - \mathbf{h} \right)$$

leads to an expression in which the total energy in the system is modified by the boundary conditions. The minimum of the expression with respect to x_i and λ_i represents the best possible solution subject to the imposed boundary conditions. Therefore,

$$\delta J = \frac{\partial J}{\partial x_i} \delta x_i + \frac{\partial J}{\partial \lambda_i} \delta \lambda_i = 0.$$

Now, $\delta\lambda_i$ and δx_i are arbitrary which implies that $\frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial \lambda_i} = 0$ since only then arbitrary values can be fulfilled. So

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{h} \end{pmatrix}.$$

This leads to the enlarged system of equations

$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ \lambda_1 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ 3 \end{pmatrix}$$

7.5.2 Neumann

Neumann boundary operators are currently not generally implemented. This implies that only a Neumann 0 boundary condition can be realised. For other Neumann boundary conditions linear triangle elements have to be employed for which non zero Neumann boundary conditions are implemented. To generally implement Neumann boundary conditions the Neumann part in the partial differential equations operators such as in Equation 6.7 have to be implemented.

7.6 Systems

The `imsSystem` data structure implements an interface for Equation 2.1. The implementation, however, for `imsSystem` is special. In principle, the implementation needed for `imsSystem` is again little more than a constructor and selector. The fact that behind the scenes `imsSystem` is a very complex implementation is to be found in the reason that it is able to provide parametric and polynomial systems, which are useful for (non-linear) model order reduction. This, however, is not scope of this presented work and the interested reader may consult other sources [9, 12].

The point is that for the presented work a simple constructor and selector is sufficient. As a matter of fact the first implementation of `imsSystem` was just that and has been subsequently changed - without the user being aware of it - to a polynomial representation.

A sample session for a system of equations is given below:

We load the package and create dummy matrices.

```
In[1]:= Needs["Imtek`System`"];
In[2]:= f = Table[{1},{3}];
In[3]:= s = d = m = Table[{1},{3},{3}];
```

We create a system and a formatted output is returned.

```
In[4]:= imsMakeSystem[f,s,d,m]
Out[4]= imsSystem[3,{1,1,1,1}]
```

The returned formatted output displays information about the contents of the system. This system has 3 degrees of freedom and has 1 load vector, 1 stiffness, 1 damping and 1 mass matrix[11].

[11]The fact that the system can carry more than one of the matrices is irrelevant for this work.

Resources:	
Implementation	System.nb\.m
Documentation	SystemDocu.nb

7.7 System Analysis

The ultimate goal for engineers is design. In order to make a good design the ability to reflect and analyse is essential. In this work we focus on how the different analysis methods interact with the `imsSystem` data structure.

7.7.1 Stationary

The stationary solution of a system of equations is the solution at $t = \infty$ when the system has - if it can - reached a stable state. A stationary solution is found by

$$u = S^{-1} f$$

which implies the use of a `LinearSolve`. The function `imsStationarySolve` is a wrapper for `LinearSolve`.

7.7.2 Non-linear

A non-linear analysis can also be performed by `imsTimeIntegrateNonlinear` or `imsStationarySolve`, which are also discussed in [9].

The main non-linear solver for this work, however, is a damped affine invariant newton solver due to [4]. Valuable details for implementation can be found in [13, 6]. Since the implementation does not wary to much from its original we do not present any details. The function is called `imsNonLinearSolve`.

Resources:	
Implementation	NonlinearSolve.nb\.m
Documentation	NonlinearSolveDocu.nb

7.7.3 Transient

`imsTimeIntegrate` does the transient, id est time dependent analysis of a system of equations and is discussed in [9]. `imsTimeIntegrate` is employed for advecting the implicit function. For the time integration of the Navier-Stokes equation we use a backward differentiation formula (BDF). We use a constant step size algorithm. Since the Navier-Stokes equation and the level-set equation are of first order we consider a first order time integration algorithm. The backward differentiation formulae have the property of integrating differential algebraic equations (DAEs) and stiff equations and is thus very well suited for our purpose [5, 2, 15, 16, 7]. We start with a first order system of equations in the spirit of 2.1

$$L = Su + D\dot{u} \tag{7.1}$$

or

$$L - Su = f(t, u) = D\dot{u}.$$

The equation is rewritten

$$\dot{u} = D^{-1}(L - Su) = \phi(t, u).$$

To derive the backward differentiation formulae we first interpolate u and then find the derivative of the interpolation. Assume that in Equation 7.1 we have solutions at times $u_{n-k+1}, \ldots, u_n$. In order to derive a formula for u_{n+1} we set up an interpolation polynomial $q(t)$ which interpolates the values $\{(t_i, u_i)|i = n-k+1, \ldots, n+1\}$. The polynomial is expressed in terms of backward differences

$$q(t) = q(t_n + sh) = \sum_{j=0}^{k}(-1)^j \begin{pmatrix} -s+1 \\ j \end{pmatrix} \nabla^j u_{n+1}.$$

The backward differences are recursively defined as

$$\nabla^0 f_n = f_n, \qquad \nabla^{j+1} f_n = \nabla^j f_n - \nabla^j f_{n-1} \ .$$

We are interested in the solution of

$$\dot{q}(t_{n+1}) = \phi(t_{n+1}, u_{n+1}).$$

Now we can rewrite

$$\frac{dq(t)}{dt} = \frac{dq(t)}{ds}\frac{ds}{dt} \Leftrightarrow \frac{dt}{ds}\frac{dq(t)}{dt} = \frac{dq(t)}{ds}.$$

With $dt/ds = h$ and keeping in mind that the differentiation is performed at $s = 1$ we have

$$h\frac{dq(t)}{dt} = h\,\phi(t_{n+1}, u_{n+1}) = \frac{dq(t)}{ds} = \frac{d}{ds}\sum_{j=0}^{k}(-1)^j \begin{pmatrix} -s+1 \\ j \end{pmatrix} \nabla^j u_{n+1}\Bigg|_{s=1} .$$

This results in

$$\sum_{j=1}^{k}\frac{1}{j}\nabla^j u_{n+1} = \sum_{j=1}^{k}\alpha_j u_{n-k+1+j} = h\,\phi_{n+1}. \tag{7.2}$$

The coefficients α_j and the order k then are

$$\begin{array}{lrcl}
k=1: & -1u_n + 1u_{n+1} & = & h\,\phi_{n+1} \\
k=2: & \frac{1}{2}u_{n-1} - 2u_n + \frac{3}{2}u_{n+1} & = & h\,\phi_{n+1} \\
k=3: & -\frac{1}{3}u_{n-2} + \frac{3}{2}u_{n-1} - 3u_n + \frac{11}{6}u_{n+1} & = & h\,\phi_{n+1} \\
k=4: & \frac{1}{4}u_{n-3} - \frac{4}{3}u_{n-2} + 3u_{n-1} - 4u_n + \frac{25}{12}u_{n+1} & = & h\,\phi_{n+1} \\
k=5: & -\frac{1}{5}u_{n-4} + \frac{5}{4}u_{n-3} - \frac{10}{3}u_{n-2} + 5u_{n-1} - 5u_n + \frac{137}{60}u_{n+1} & = & h\,\phi_{n+1}.
\end{array}$$

Two steps remain. First, we must account for the damping matrix D and secondly, rewrite Equation 7.2 such that it is suitable for the non-linear solver. In each step we pre-evaluate

$$\text{tail} = \frac{1}{\alpha_k}\sum_{j=1}^{k-1}\alpha_j u_{n-k+1+j}$$

and set up the residual function

$$g(u_{n+1}) := D(t_{n+1}, u_{n+1}) \cdot (u_{n+1} + \text{tail}) - \frac{h}{\alpha_k} f(t_{n+1}, u_{n+1})$$

and the Jacobian

$$m(u_{n+1}) := D(t_{n+1}, u_{n+1}) - \frac{h}{\alpha_k} J(t_{n+1}, u_{n+1}).$$

The implementation of the algorithm is thus given as:

```
imsBDFStep[f_,jacobian_,mass_,oldVals_,currentTime_,dt_,coeffs_,opts___?OptionQ ]
   := Module[{bk = 1/coeffs[[-1]],g,m,tail},
   tail = (Plus @@ Times[Most[coeffs],oldVals])/coeffs[[-1]];
   g[un_List] := mass[currentTime+dt,un].(un+tail)-dt*bk*f[currentTime+dt,un];
   m[un_List] := mass[currentTime+dt,un]-dt*bk*jacobian[currentTime+dt,un];
   imsNonlinearSolve[m,g,oldVals[[-1]],opts] ]
```

With a few more lines of code we can implement the solver which continuously applies the `imsBDFStep`.

```
imsBDFSolve[f_,j_,m_,oldVals_,{tStart_,tEnd_,dt_},coeffs_,opts___?OptionQ ] :=
   Block[{thisVals = RotateRight[oldVals]},
   {Partition[Join[#,{tEnd}],1],
    ToPackedArray[FoldList[
     imsBDFStep[f,j,m,thisVals=Rest[Join[thisVals,{#1}]];
      thisVals,#2,dt,coeffs,opts]&,Last[oldVals],#]]}&[Range[tStart,tEnd-dt,dt]]]
```

7.7.4 Harmonic

In harmonic analysis the representation of functions as superposition of basic waves - harmonics - is studied. After a harmonic analysis we can draw conclusions about the filter characteristics of the system. The system may expose low-pass, high-pass, or band-pass characteristics.

The function `imsHarmonicSolve` does a harmonic analysis for the Laplace[12] transformed of a system. For first order systems this solves in effect

$$s = (\imath\,\omega D + S)^{-1} f.$$

For second order systems

$$s = \left(-\omega^2 M + i\,\omega D + S\right)^{-1} f$$

is solved. As input the user must specify the frequencies g of interest which are then transformed via

$$\omega = 2\pi g.$$

7.7.5 Eigenwert

For the Eigenwert computation we use the Mathematica internal function `Eigensystem`.

Resources:	
Implementation	SystemAnalysis.nb\.m
	TimeIntegrate.nb\.m
Documentation	SystemAnalysisDocu.nb
	TimeIntegrateDocu.nb

References

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. The MIT Press, 2nd edition, 1996.

[2] Uri M. Ascher and Linda R. Petzold. *Computer Methofs for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[3] R. Barrere. An algorithmic approach to manifolds. *The Mathematica Journal*, 11(1):To Appear, To Appear.

[12] Although the Laplace transform is named in honour of Laplace, who used the transform in his work on probability theory, the transform was discovered originally by Leonhard Euler.

[4] Peter Deuflhard. A modified newton method for the solution of ill-conditioned systems of nonlinear equations with appication to multiple shooting. *Numerische Mathematik*, 22:289 – 315, 1974.

[5] C.W. Gear and L.F. Shampine. A user's view of solving stiff ordinary differential equations. *SIAM Review*, 21, No.1:1 – 17, 1979.

[6] Michael Hanke. Towards a new implementation of a nonlinear solver for femlab. Technical report, Stockholm: Royal Inst. of Technology, Report TRITA-NA-0011, 2000.

[7] Michael Hanke. A new implementation of a bdf method within the method of lines. *J. Comput. Meth. Sci. Eng.*, To appear 2003.

[8] Victor N Kaliakin. *Introduction To Approximate Solutions Techniques, Numerical Modeling, and Finite Element Methods*. Marcel Dekker, 2002.

[9] Jan Lienemann. *Complexity Reduction Techniques for Advanced MEMS Actuators Simulation*. PhD thesis, University Freiburg, 2006.

[10] Roman E. Maeder. *Programming in Mathematica*. Addison-Wesley, 3rd edition, 1997.

[11] Roman E. Maeder. *Computer Science with Mathematica*. Cambridge University Press, 2000.

[12] Christian Moosmann. *ParaMOR - Model Oder Reduction for parameterized MEMS applications*. PhD thesis, University of Freiburg, 2007.

[13] Nowak and Weimann. A family of newton codes for systems of highly nonlinear equations. *Konrad-Zuse-Zentrum*, Technical Report TR-91-10:na, 1991.

[14] O. Rübenkönig and J. G. Korvink. Imtek mathematica supplement. `http://www.imtek.uni-freiburg.de/simulation/mathematica/IMSweb/`, 2002 – 2007.

[15] G. Wanner and E. Hairer. *Solving Ordinary Differential Equations I Nonstiff Problems*. Springer-Verlag, 1993.

[16] G. Wanner and E. Hairer. *Solving Ordinary Differential Equations II Stiff and Differential-Algebraic Problems*. Springer, 1996.

8 Application Examples

In mathematics you don't understand things.
You just get used to them.
(von Neumann)

8.1 Verification of the Code **120**
8.2 Test Cases **122**
8.3 The Ink-Jet Model **135**
8.4 Final Remarks **138**

POSSIBLE scientific and engineering duties that can be solved with IMS are numerous; and it is clear that not all can be presented here. We restricted ourselves to show the results concerning the free surface flow. First, however, some measurement and convergence analysis techniques are introduced. These techniques are then, in a second step, applied to free surface flow with increasing complexity. We start with a pure advection of a circular bubble. Next we examine the behaviour of bubbles in different flow regimes. In the last section we show the simulation of an ink-jet print head, taking into account the ejection, flight and impact on a target of the liquid droplet.

8.1 Verification of the Code

8.1.1 How to measure...

In this section we verify the program code. It is necessary to describe the mathematical tools employed and to define the anticipated results. Unfortunately, very few analytical results are available for free surface flow. To remedy this short coming we compare to analytical test problems where possible and rely on numerical measurements of other colleagues where no analytical results are available.

8.1.1.1 Errors, precision & accuracy

If we have an exact solution denoted by $\mathbf{u}_e$ we measure two kinds of errors. First the absolute error: $\mathbf{e}_a = \mathbf{u}_e - \mathbf{u}$ where $\mathbf{u}$ is the approximate solution. Secondly, we have the relative error: $\mathbf{e}_r = (\mathbf{u}_e - \mathbf{u})/\mathbf{u}_e$. The relative error can be represented as a percentage by multiplying the relative error by 100. Also, if an approximate value has a relative error of about 10^{-p}, then its decimal representation has about p correct significant digits (significant digits are the leading nonzero digits and all following digits). The relative error thus refers to the accuracy of the number. The precision of a number refers to the amount of digits necessary to express a number [6].

8.1.1.2 Convergence rate

An iterative method is said to converge with rate r if

$$\lim_{k\to\infty} \frac{||\mathbf{e}_{k+1}||}{||\mathbf{e}_k||^r} = C$$

for some finite constant $C > 0$. Here $\mathbf{e}_k$ denotes the error at iteration k [6].

8.1.1.3 Norms

For measurements we use the p-norm or variants of it. The p-norm is defined as

$$||\mathbf{u}||_p = \left(\sum_{i=1}^{M} |\mathbf{u}_i|^p \right)^{1/p} .$$

8.1.2 ...and what to expect

8.1.2.1 Spacial integration

For spacial integration we use finite elements. As a rule of thumb it holds that finite element shape functions of order q have an order of accuracy of $q+1$.

Example: We use second order elements for velocity and first order elements for pressure. The accuracy for the velocity behaves proportional to h^{q+1}. If we half the mesh size h in each space direction ($\approx$ four times the number of elements in 2-dimensions) we expect $(h/2)^{2+1} = 1/8\,h^3$. The accuracy should thus be 8 times better. For the linear pressure elements we expect an improvement of $(h/2)^{1+1} = 1/4\,h^2$, which corresponds to a 4 times better accuracy.

8.1.2.2 Non-linear iteration

For the rate of convergence for Newton's method it holds that [10]:

$$||\mathbf{u}_{k+1} - \mathbf{u}_e||_\infty \leq C||\mathbf{u}_k - \mathbf{u}_e||_\infty^2 \text{ , with } ||\mathbf{u}_0 - \mathbf{u}_e||_\infty \leq \epsilon.$$

C is a constant. The quadratic convergence only holds if the initial guess $\mathbf{u}_0$ to the exact solution $\mathbf{u}_e$ is in the convergence region ϵ. In other words: the number of correct digits is roughly doubled at each iteration of Newton's method [6]. For mesh based non-linear iterations it is useful to use the root mean square (RMS) norm [9]

$$\|\cdot\|_{rms} = \frac{1}{N}\left(\sum^N u_i^2\right)^{1/2}.$$

8.1.2.3 Time integration

Once the spacial discretisation is done a backward differentiation formulae (BDF) solver is appointed for integrating the resultant ordinary differential equation. In the case of the Navier-Stokes equation we deal with a system of differential algebraic equations (DAE). Good references on the topic of BDF and DAE are given by Gear and others [4, 1, 15, 5].

The order of accuracy of a BDF of order q is proportional to $\triangle t^q$. Half the time step size $(\triangle t)/2$ thus leads to $((\triangle t)/2)^q$ improvement in the accuracy of the result.

Example: We use a second order BDF. Thus $q = 2$. If follows that $((\triangle t)/2)^2 = 1/4\ (\triangle t)^2$. The accuracy should improve by a factor of 4 if the time step size is halved.

8.2 Test Cases

The free surface algorithm presented allows for setting the frequency at which the implicit function ϕ is re-initialised (see 6.3.6.2 on page 83). For better comparability we have chosen to do full re-initialisation after each time step for all presented test cases. Also the correction of the implicit function is done after each time step. The most delicate issue is the good choice of the interface width h_w introduced on page 84. It takes a lot of experience and patience to find suitable initial setting. As a starting point an interface thickness of about 3.5 elements to each side of the interface is chosen.

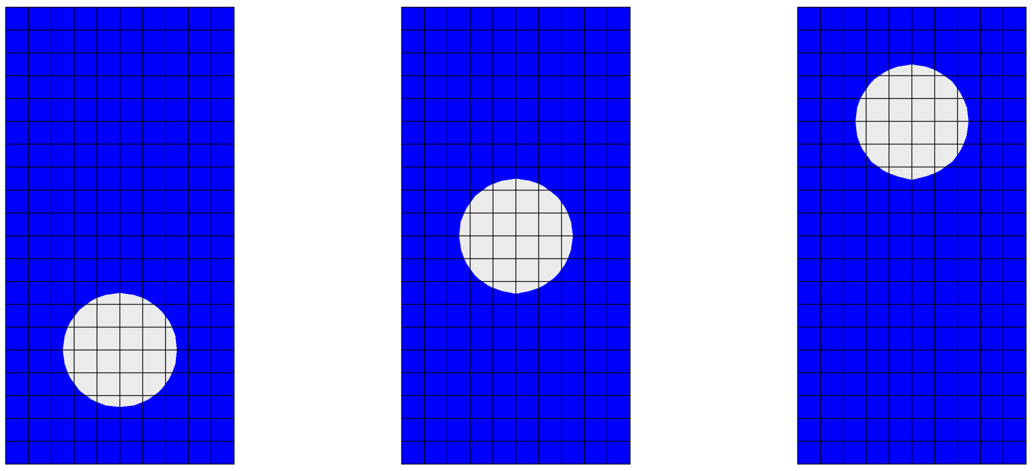
8.2.1 Advecting a cylindrical bubble in a tube

In the first test case we investigate the pure advection of a cylindrical bubble [8]. The purpose of this test case is to establish whether a one-directional coupling from the Navier-Stokes solver to the level-set works on a very simple level. The surface tension is neglected and thus the position of the interface does not influence the fluid flow. In other words, the fluid flow is decoupled from the position of the interface. Since there is no interaction from the free surface term to the fluid we have a quasi analytical solution. Furthermore, the parameters of the fluids are chosen to be not very demanding on the Navier-Stokes solver. The solver will thus find the correct flow field quickly and with high precision and it is not an issue from which hassle in respect to the level-set method is to be expected. In this case the flow field is $1\,\mathrm{m/s}$ and pointing upward. It is this flow flied that is plugged into the level-set method to advect the circular bubble. This setup provides an analytical solution: after $1\,\mathrm{s}$ the bubble is advected $1\,\mathrm{m}$ and no change to the bubble's form can occur.

The settings for the test problem are summarised as follows:

$$\rho_1 = \rho_2 = 1\,\mathrm{kg/m^3},\ \mu_1 = \mu_2 = 1\,\mathrm{Pa\,s},\ \sigma = 0\,\mathrm{N/m},\ g = 0\,\mathrm{m/s^2}.$$

The dynamic viscosity μ and density ρ are set to 1. The surface tension σ is set to 0 and thus the interface width h_w needs not to be set. Gravity g is neglected and set to 0. The spacial domain is $\Omega \in [0,2] \times [0,4]$. On the vertical walls we have a slip boundary condition. On the bottom we have an inflow boundary condition with $u = 0\,\mathrm{m/s}$ and $v = 1\,\mathrm{m/s}$. At the top the pressure boundary condition p is set to 0. A simulation is performed from time $t_0 = 0\,\mathrm{s}$ until $t_e = 2\,\mathrm{s}$ in a step size of $\triangle t = 0.25\,\mathrm{s}$. The bubble radius is set to $R = 1/2$ meters and its centre's initial location is at $(1,1)$. Figure 8.1 shows the circular bubble at different time steps. In this case the bubble is advected 2 meters, which is exactly the behaviour that was expected. Also the bubble's shape is conserved.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 8.1: To test the interaction between the Navier-Stokes solver and the level-set method on a fundamental level a circular bubble is placed into an upward flow field. The geometry and the physical data are selected in such a manner that the resulting flow field is $\mathbf{u} = (0, 1)\,{}^{\mathrm{m}}/_{\mathrm{s}}$ upwards. Since the buoyancy force is neglected the flow field is decoupled from the position of the bubble. The figures show from left to right the bubble at time steps $t = 0, 1, 2\,\mathrm{s}$. The form of the bubble remains unchanged. The grid lines are for measurement purposes only. The actual computation hast been done with 1024 triangular elements.

8.2.2 Static bubble - Testing surface tension

With the following experiment we will investigate the working of the surface tension term. For this we set up a circular static bubble in equilibrium. The bubble is situated in a liquid. This experiment is based on the Laplace-Young law which relates a pressure difference of two static fluids to the surface tension and the radius of the bubble. The pressure outside of the bubble p_{out}, that is the pressure in the liquid, is assumed constant, as is the radius R of the bubble. For a given value of the surface tension σ we examine the pressure p_{in} inside the bubble. The experiment is repeated on different meshes and with different interface thicknesses h_w. The velocity field $\mathbf{u}$ of the fluids is zero every where. This test was considered by Lafauriel [7] and later by Smolianski [13]. The analytical solution is summarised as:

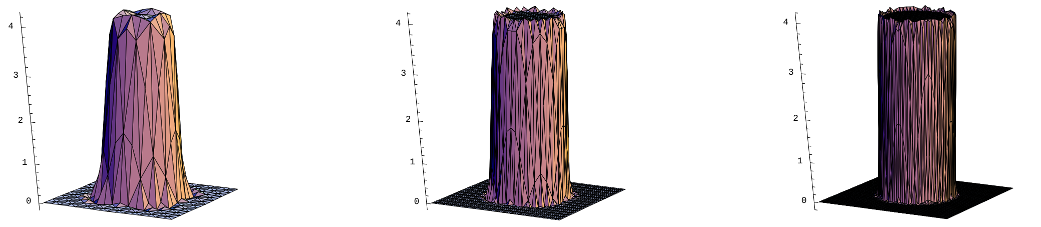
$$\begin{aligned} \mathbf{u} &= \mathbf{0} \\ p &= \begin{cases} p_{in} = p_{out} + \frac{\sigma}{R} & \text{if } x^2 + y^2 \leq R^2 \\ p_{out} & \text{else} \end{cases} . \end{aligned}$$

The simulation domain is $\Omega \in [0, 1]^2$. The radius of the bubble is set to $R =$

$1/4$ meters and its centre is set to $(1/2, 1/2)$. At this point the pressure p_5 is measured. The outside pressure p_{out} is specified somewhere on a point of the domain border. The physical parameters of the experiment are set up as follows:

$$\rho_1 = \rho_2 = 1\,{}^{\mathrm{kg}}\!/\!{}_{\mathrm{m}^3},\ \mu_1 = \mu_2 = 1\,\mathrm{Pa\,s},\ \sigma = 1\,{}^{\mathrm{N}}\!/\!{}_{\mathrm{m}},\ g = 0\,{}^{\mathrm{m}}\!/\!{}_{\mathrm{s}^2}.$$

The dynamic viscosity μ and density ρ are set to 1. The surface tension σ is set to 1. Gravity g is neglected and set to 0. On the boundary ($\partial\Omega$) we set no slip conditions for the velocity components (i.e. $\mathbf{u} = 0\,{}^{\mathrm{m}}\!/\!{}_{\mathrm{s}}$) and one pressure point ($\partial\partial\Omega^1$) is set. (i.e. $p_{out} = 0$). The pressure inside the bubble can then be expected to be $p_{in} = 0 + 1/1/4 = 4\,\mathrm{Pa}$. The computations are performed until $t_e = 5\,\mathrm{s}$ in steps of $\triangle t = 0.05\,\mathrm{s}$. The resultant pressure on different meshes is depicted in Figure 8.2.



(a) 512 elements and $h_w = 0.11$

(b) 2048 elements and $h_w = 0.11/2$

(c) 8192 elements and $h_w = 0.11/4$

Figure 8.2: To test the surface tension a bubble is placed into a liquid. The Laplace-Young law relates a pressure difference of two static fluids to the surface tension σ and the radius R of the bubble. $p_{in} = p_{out} + \sigma/R$. Here, the outside pressure p_{out} is set to zero. The surface tension σ is set to one and the radius is set to $1/4$ meters. The figures show the pressure p_{in} inside the static bubble for three different interface width h_w and three different mesh sizes.

We conduct the same experiment with three different meshes. In order to estimate the convergence we use 512, 2048 and 8196 elements. This increases the number of elements by a factor of 4 - which corresponds to a factor of 2 in each space direction. At the same time we decrease the interface width h_w by a factor of 2 starting from $h_w = 0.11$. In the experiment based on the fine grid we do an additional fourth experiment where the interface thickness is not further divided and set to $h_w = 0.11/2$. To asses the free surface algorithm's computational complexity we first measure the error of the velocity part of the solution

[1]If $\partial\Omega$ indicates the boundary of a domain, then $\partial\partial\Omega$ indicates a part of the boundary. In two dimensions the boundary $\partial\Omega$ is a line and $\partial\partial\Omega$ indicates a point on that line.

vector in an averaged norm. The velocity norm used is

$$\|\cdot\|_1 = \frac{1}{N}\sum_{i=1}^{N} |u_i| \, .$$

Also of interest are the absolute and relative error of the pressure. Last, we also give the pressure p_5 measured at the centre point $(0.5, 0.5)$ of the simulation domain. The discussed results are summarised in Table 8.1.

Table 8.1: The table presents the order of accuracy measurements for the static bubble experiment. The accuracy depends on the number of elements and the interface thickness h_w. First the error in the velocity is listed. Then the absolute and relative errors of the pressure, based on the maximum pressure value found in the simulation domain. In the last column we list pressure p_5 at the point $(0.5, 0.5)$.

h^2	h_w	$\|\|\mathbf{u} - \mathbf{u}_e\|\|_1$	$\|p_{in} - p_{out} - \frac{\sigma}{R}\|$	$\frac{\|p_{in}-p_{out}-\frac{\sigma}{R}\|}{\frac{\sigma}{R}} \times 100\%$	p_5
1/512	0.11	0.00030945	0.18	4.5	4.13116
1/2048	0.11/2	0.0000895176	0.11	2.75	4.02749
1/8192	0.11/2	0.0000225063	0.045	1.125	4.0256
1/8192	0.11/4	0.0000336096	0.098	2.45	4.00652

First we consider the scenario where the sequence of the interface thickness is set to $h_w = 0.11$, $0.11/2$ and $0.11/2$. For the velocity we find that order q in the discrete L_1 ("average") norm is

$$\begin{aligned} q = \sqrt{\frac{0.00030945}{0.0000895176}} &= 1.86 \\ q = \sqrt{\frac{0.0000895176}{0.0000225063}} &= 1.99. \end{aligned}$$

We find that despite the pressure discontinuity the pressure norm exhibits about first order convergence. This test shows that the surface tension is modelled satisfactorily. At the interface in Figure 8.2 we see overshoots and undershoots which mainly contribute to the absolute and relative pressure measured.

If we consider the sequence in which the interface thickness is set to $h_w = 0.11$, $0.11/2$ and $0.11/4$ we note that the pressure value p_5 on the plateau is decaying well. This, however comes at the cost of a diminished order of convergence for the velocity and absolute pressure. We conclude that the interface thickness can not be diminished indefinitely without affecting velocity and pressure. The author is not aware of any rational to estimate the best interface thickness for a given problem. It is noteworthy that the maximum and minimum pressure which are a measure for the overshoot do decrease for the first set of the interface width.

The algorithm presented is, however, not as proficient as the results by Smolianski [13]. This, however, is due to the fact that we take a holistic approach in the sense that no special treatment such as subdivision of the elements containing the free surface is needed. From this perspective our code is more general but at a performance cost.

The experiment's resultant velocity field should be zero everywhere. The measured off zero velocity are called spurious currents and are observed close to the free surface interface as shown in Figure 8.3. The amplitude of the spurious currents is proportional to σ/μ [13]. Currently at least two causes are identified to account for the formation of spurious currents [14]: The smoothing of the discontinuity (the surface tension) can be detrimental. The mass in a cell is accelerated by the body force. In the transition region the body force and the mass do not vary at the same rate. While the mass approaches a small value at the interface the body force my still be relatively large. A second course for spurious currents can be seen in the numerical imbalance between the surface tension force and the associated pressure gradient.

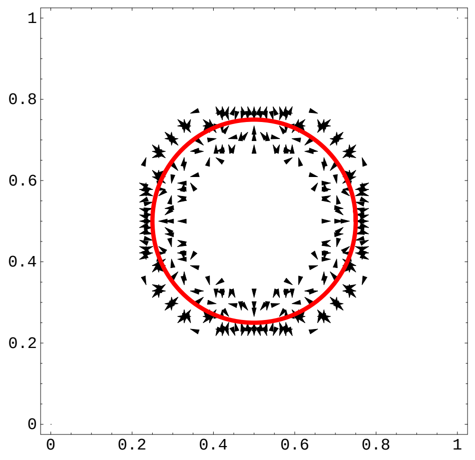


Figure 8.3: To treat the discontinuity between to fluids the interface region is smoothed over. The smoothing of the discontinuity results in spurious currents which can be seen at the interface in a static bubble simulation. The mesh consisted of 2048 elements.

8.2.3 Rising bubble in tube - Buoyancy

In this experiment the free surface algorithm is employed to reproduce bubble behaviour as it is observed by laboratory experiments. The behaviour of the bubble is modified through the physical parameters of the fluids such as the viscosity, density and the surface tension. Since we have examined the surface tension in a previous experiment, we now examine the code's capability to handle different density and viscosity values. To that end we examine the rising of bubbles. The physical properties of the liquids and bubbles are adjusted in such a manner that different physiological behaviours of the bubble motion can be observed. In Figure 8.4 (reproduced with kind permission of Prof. Clift [2]) different Reynolds[2] numbers and Eötvös[3] numbers are depicted on the axes. Depending on the constellation the rising bubble is more or less ellipsoidal or skirted. Three tests have been conducted and the expected bubble behaviour could be reproduced. An additional forth experiment was carried out which shows good agreement with the results from another group [8].

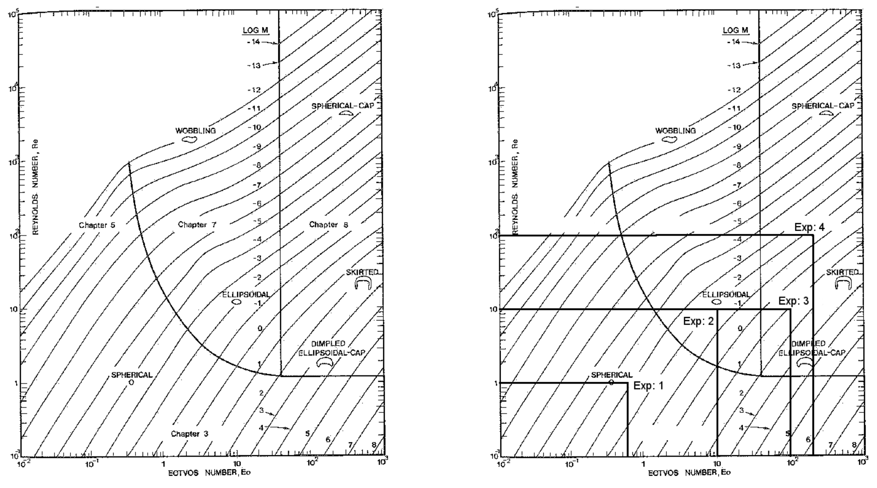


Figure 8.4: To the left: Experimental data displaying different regimes of bubble and droplet shapes plotted against Reynolds and Eötvös numbers. (The figure was reproduced with kind permission of Prof. Clift [2]). To the right: The same figure again, this time indicating four numerical experiments (Exp 1 to 4) conducted. We found that the expected bubble shapes, depicted at the intersection of the horizontal and vertical lines, attributed to the different Reynolds and Eötvös number could be reproduced.

In the conducted experiments we set

$$\begin{aligned} \rho_1 &= 1000\,{}^{\mathrm{kg}}/_{\mathrm{m}^3}, & \rho_2 &= 1\,{}^{\mathrm{kg}}/_{\mathrm{m}^3} \\ \mu_1 &= 10^{-3}\,\mathrm{Pa\,s}, & \mu_2 &= 10^{-5}\,\mathrm{Pa\,s} \\ & & g &= 9.8\,{}^{\mathrm{m}}/_{\mathrm{s}^2}. \end{aligned}$$

[2]The Reynolds number is a dimensionless ratio of the inertial forces to the viscous forces in the fluid. The ratio is given as $Re = {}^{\rho UL}/_{\mu}$. Here ρ is the fluid's density in ${}^{\mathrm{kg}}/_{\mathrm{m}^3}$ and μ is the fluid's dynamic viscosity in $\mathrm{Pa\,s}$. U is a typical velocity $\mathrm{m}s$ and L is a typical length in m.

[3]The Eötvös number is a dimensionless ratio of body forces to surface tension. The ratio is given as $Eo = {}^{\triangle\rho g L^2}/_{\sigma}$. Here $\triangle\rho$ is the fluids' density difference in ${}^{\mathrm{kg}}/_{\mathrm{m}^3}$. g is the gravitational acceleration in ${}^{\mathrm{m}}/_{\mathrm{s}^2}$. L is a typical length in m and σ is the surface tension in ${}^{\mathrm{N}}/_{\mathrm{m}}$.

Via the Reynolds number - which is given - we compute the typical length L and set the initial radius r of the cylindrical bubble to it. The width of the simulation domain is four times the typical radius. The height is eight times the typical radius. Next, we set the surface tension. This is done by employing the Eötvös number; which is again given. The interface width was set to $h_w = 0.1 \times 4 \times r$. The computation was performed on a totally irregular triangle mesh with various number of elements. The simulation time was from $t_0 = 0$ seconds to $t_e = 0.03$ seconds in steps of $\triangle t = 0.00025$ seconds. We now consider the test cases in more detail.

8.2.3.1 Cylindrical bubble

The first experiment is to reproduce a cylindrical bubble. In this case the shape of the bubble does not change. The bubble is advected. In contrast to the previous experiment, however, the coupling between the fluid flow and the free surface is in both directions. To setup the experiment we set the following parameters:

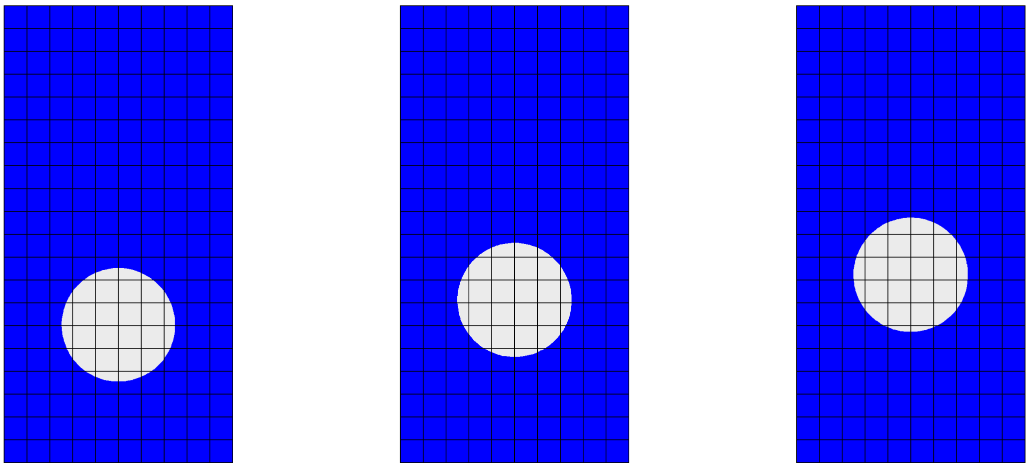
$$\begin{aligned} Re = 1 = \frac{(2r)^{3/2}\sqrt{g}\rho_1}{\mu_1} \quad &\rightarrow \quad r = \frac{1}{2}\left(\frac{\mu_1 Re}{\sqrt{g}\rho_1}\right)^{2/3} = 0.000023365 \\ Eo = 0.6 = \frac{4\rho_1 g r^2}{\sigma} \quad &\rightarrow \quad \sigma = \frac{4\rho_1 g r^2}{Eo} = 0.00003567. \end{aligned}$$

The mesh consists of 6335 elements which corresponds to 41906 degrees of freedom. Figure 8.5 shows the results and Figure 8.8 shows the interface length and the amount of re-initialisation done during the simulation.

The circumference of the circle can be computed to be $2r\pi = 0.000146807$ meters. In our measurement we find the circumference to be on average 0.0001468 meters. Which is an absolute error of about $6.6 \cdot 10^{-9}$ and a relative error of about 0.0045%. The amount of each re-initialisation is about $d = \pm 2 \cdot 10^{-8}$. Since the implicit surface is a signed distance function we thus know that the radius of the bubble varies by the amount the implicit surface is raised or lowered. The exact area of the cylindrical bubble is $\pi r^2 = 1.71507 \cdot 10^{-9}$. The variation in the radius modifies the area by $\pi(r + d)^2$. The relative variation in the bubble's area is then about 0.17%. We also find that the numerical result agrees well with the findings of Clift [2].

8.2.3.2 Ellipsoidal bubble

In the second experiment we which to model a bubble in the ellipsoidal bubble regime. To this end we set the bubbles radius r and the surface tension σ to the



(a) $t = 0.01$ (b) $t = 0.02$ (c) $t = 0.03$

Figure 8.5: Experiment 1: The circular bubble in a tube is rising due to the buoyancy force. The geometry and physical data have been set up to correspond to a Reynolds number $Re = 1$ and an Eötvös number $Eo = 0.6$. In this flow regime no change of the bubble shape is expected. We have a cylindrical bubble at the three different times $t = 0.01$, 0.02, 0.03 seconds.

following parameters:

$$Re = 10 = \frac{(2r)^{3/2}\sqrt{g}\rho_1}{\mu_1} \quad \longrightarrow \quad r = \frac{1}{2}\left(\frac{\mu_1 Re}{\sqrt{g}\rho_1}\right)^{2/3} = 0.00010845$$

$$Eo = 10 = \frac{4\rho_1 g r^2}{\sigma} \quad \longrightarrow \quad \sigma = \frac{4\rho_1 g r^2}{Eo} = 4.61044\,10^{-5}$$

The mesh consists of 6749 elements which corresponds to 44590 degrees of freedom. Figure 8.6 shows the results and Figure 8.8 shows the interface length and the amount of re-initialisation which are given for completeness. In this case it is not possible to conduct an estimation of the relative error based on the interface length and the amount of re-initialisation. We find that the numerical result agrees well with the findings of Clift [2] as can be seen from Figure 8.4.

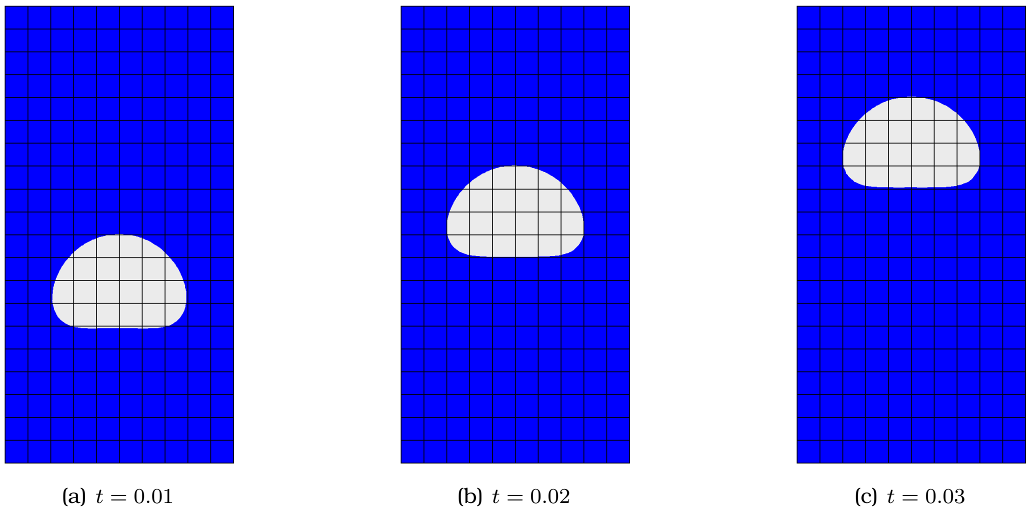

(a) $t = 0.01$ (b) $t = 0.02$ (c) $t = 0.03$

Figure 8.6: Experiment 2: A circular bubble in a tube is rising due to the buoyancy force. The geometry and physical data have been set up to correspond to a Reynolds number $Re = 10$ and an Eötvös number $Eo = 10$. In this flow regime a change of the initially circular bubble shape to an ellipsoidal shape is expected. From left to right: We show the ellipsoidal bubble shape at the three different times $t = 0.01, 0.02, 0.03$ measured in seconds.

8.2.3.3 Skirted bubble

To observe a truly skirted bubble we set the following parameters:

$$Re = 10 = \frac{(2r)^{3/2}\sqrt{g}\rho_1}{\mu_1} \quad \rightarrow \quad r = \frac{1}{2}\left(\frac{\mu_1 Re}{\sqrt{g}\rho_1}\right)^{2/3} = 0.00010845$$

$$Eo = 100 = \frac{4\rho_1 g r^2}{\sigma} \quad \rightarrow \quad \sigma = \frac{4\rho_1 g r^2}{Eo} = 4.61044 \cdot 10^{-6}$$

The mesh consists of 7254 element which corresponds to 47911 degrees of freedom. Figure 8.7 shows the results. We find a skirted bubble as it is in accordance with the laboratory experiments in Figure 8.4. Figure 8.8 shows the interface length and the amount of re-initialisation, which in this case are again given for the sake of completeness. We find that the numerical result agrees well with the findings of Clift [2]. As can be seen from Figure 8.4 for the settings given above we are in between the dimpled ellipsoidal-cap regime and the highly skirted bubble regime.

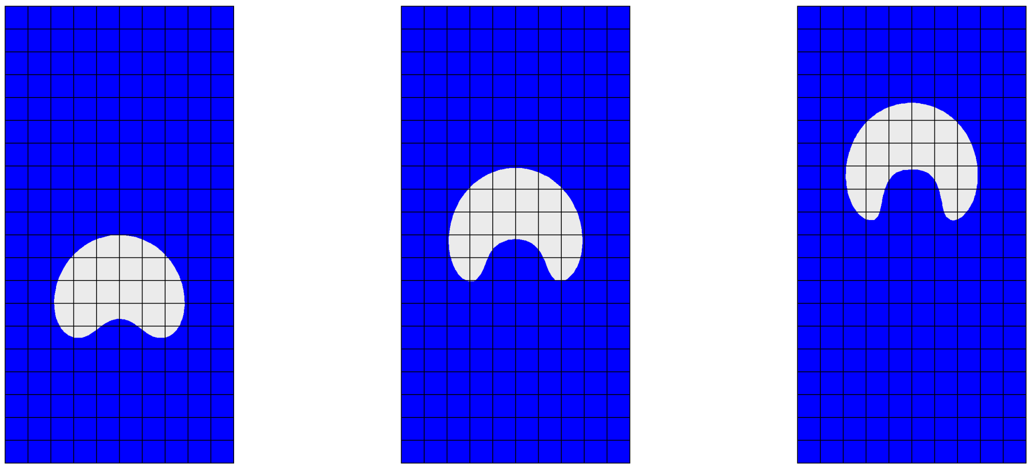


Figure 8.7: Experiment 3: A circular bubble in a tube is rising due to the buoyancy force. The geometry and physical data have been set up to correspond to a Reynolds number $Re = 10$ and an Eötvös number $Eo = 100$. In this flow regime a change of the initially circular bubble shape to a skirted shape is expected. From left to right: We show the increasingly skirted bubble shape at the three different times $t = 0.01, 0.02, 0.03$ measured in seconds.

8.2.3.4 Crosschecking with other numerical results

In the fourth experiment of this series we examine the accordance of our free surface algorithm with that of another group's [8]. In this experiment we base our experiment on the physical parameters of water (index 1) and air (index 2). We set the following parameters:

$$\begin{aligned} \rho_1 &= 1000\,{}^{\mathrm{kg}}/_{\mathrm{m}^3}, & & \rho_2 = 1.226\,{}^{\mathrm{kg}}/_{\mathrm{m}^3} \\ \mu_1 &= 0.35\,\mathrm{Pa\,s}, & & \mu_2 = 3.58 \cdot 10^{-3}\,\mathrm{Pa\,s} \\ r &= 0.025\,\mathrm{m}, & \sigma = 0.125\,{}^{\mathrm{N}}/_{\mathrm{m}}, & \; g = 10\,{}^{\mathrm{m}}/_{\mathrm{s}^2}. \end{aligned}$$

Between the two phases we have a density ratio of approximately 1000 and the viscosity ratio is about 100. The settings result in a Reynolds number of approximately $Re \approx 100$ and an Eötvös number of $Eo = 200$. The geometry of the tube is $0.1\mathrm{m}$ wide and $0.35\mathrm{m}$ high. A constant pressure boundary condition is applied at the outflow and a zero velocity was assigned at the bottom. The bubble then starts from rest. The pictures in Figure 8.9 show snapshots of the solution at time $t = 0.05\,\mathrm{s}$, $t = 0.15\,\mathrm{s}$ and $t = 0.3\,\mathrm{s}$, respectively. The computation was performed in time steps of size $\triangle t = 0.001\,\mathrm{s}$. The interface width was set to $h_w = 0.01$ and the implicit function was fully re-initialised after each time step. The mesh

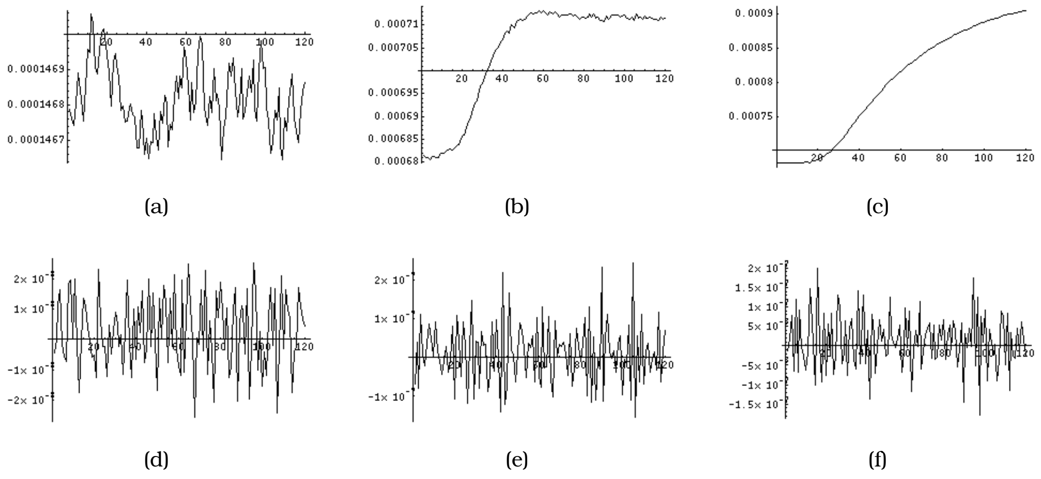


Figure 8.8: Comparing the interface length and the amount of re-initialisation of the level set function for three different bubble types. The first row depicts the interface length. The second row shows the amount of re-initialisation. The columns are for the bubbles were $Re = 1$ and $Eo = 0.6$, $Re = 10$ and $Eo = 10$ and $Re = 10$ and $Eo = 100$, respectively. On the top left we have a bubble not changing it's shape significantly (experiment 1). The top middle bubble evolves from a circular bubble to an ellipsoidal bubble. It can be seen that this process reaches an equilibrium. The top right bubble shows the development of a skirted bubble. Here the interface length is asymptotically increasing.

consisted of 5887 elements and 20887 degrees of freedom.

The bubble begins to rise due to the difference in density of the two fluids. The rising induces a vortex. The flow field from the top of the bubble divides and passes down again besides the bubble along the wall in negative y-direction. Below the bubble we see a liquid jet pushing the centre of the bubble up. First the bubble develops a skirted behaviour. With time, the down pull at the side of the tube tears the bubble apart. The centre part of the bubble continuous it's upward movement. The left and right broken off satellite bubbles are pulled in a downwards direction. In this setting the cap of the bubble is not pierced. To completely resolve the shredded bubbles a very fine mesh is required. The solutions found are in good agreement with Nagrath [8] and fit also well into Figure 8.4.

8.2.4 Rising bubble bursting

In the following example we examine the rising of an oil bubble bursting through a oil-water layer. This experiment examines the algorithm's capability to handle merging of liquids. The oil (index 2) bubble is situated in water (index 1) and

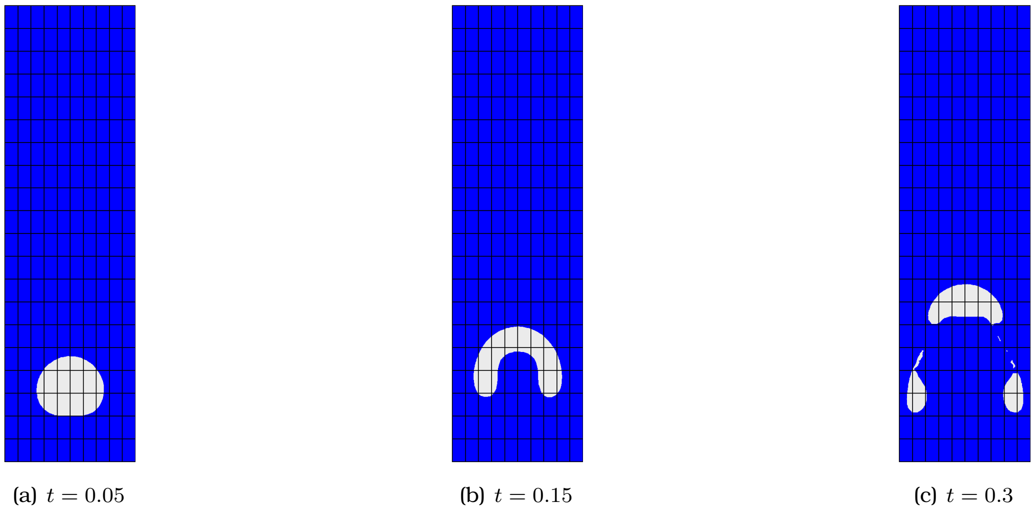

(a) $t = 0.05$ (b) $t = 0.15$ (c) $t = 0.3$

Figure 8.9: Evolution of rising bubble at different times as presented in Nagrath [8]. The bubble is teared apart by both the pull down at the side of the tube and flow field in the middle part of the tube that pushes the centre part of the bubble in an upward direction.

will burst through the surface and merge with another layer of oil. The physical parameters are as follows:

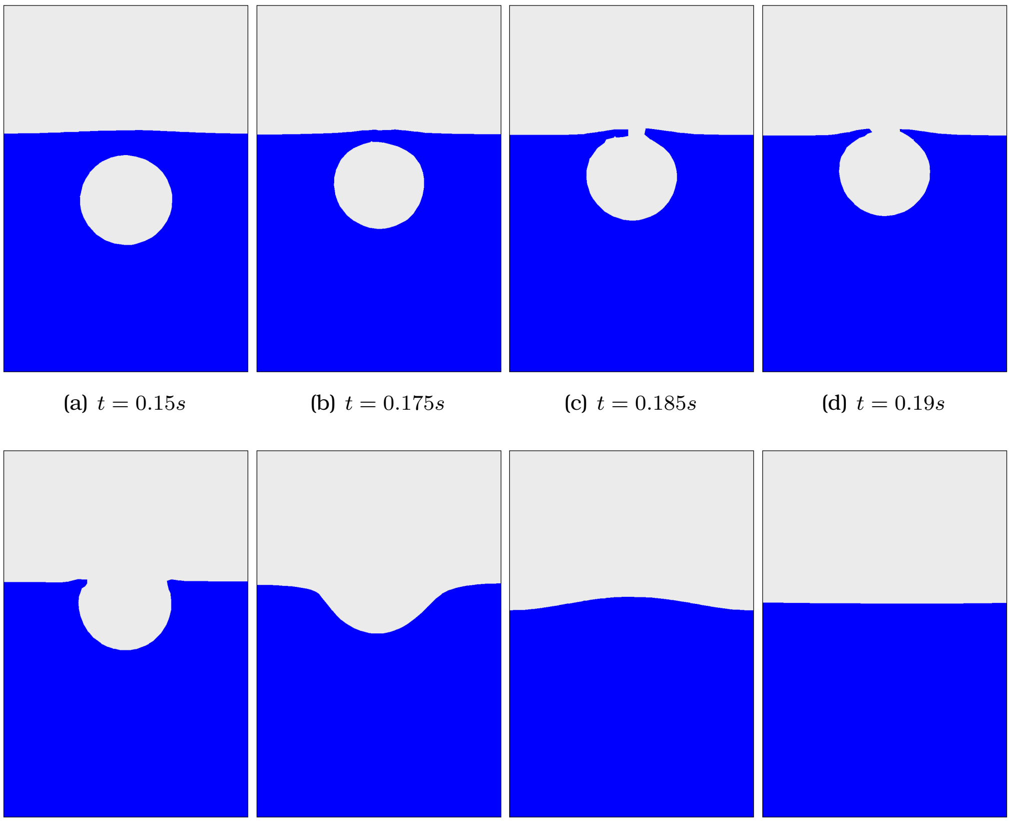
$$\begin{aligned} \rho_1 &= 997\,{}^{\mathrm{kg}}/_{\mathrm{m}^3}, & \rho_2 &= 880\,{}^{\mathrm{kg}}/_{\mathrm{m}^3} \\ \mu_1 &= 1.04\cdot 10^{-3}\,\mathrm{Pa\,s}, & \mu_2 &= 0.36\,\mathrm{Pa\,s} \\ \sigma &= 0.073\,{}^{\mathrm{N}}/_{\mathrm{m}}, & g &= 9.82\,{}^{\mathrm{m}}/_{\mathrm{s}^2}. \end{aligned}$$

The computational domain Ω is a container and set to be $[-0.005, 0.005] \times [0, 0.015]$ meters. The mesh consists of 2486 elements which corresponds to 16576 degrees of freedom. The radius of the initial bubble was set to $r = 0.002\,\mathrm{m}$ at $(0, 0.05)$. So that the initial implicit function ϕ is composed of the minimum of an implicit function for a plane and an implicit function for a circle:

$$\phi(x, y) = \min(-(y - 0.01),\ \sqrt{x^2 - (y - 0.005)^2} - 0.002).$$

We set the interface width $h_w = 0.001$ and the re-initialisation is fully done in every time step. The simulation was performed from $t = 0$ seconds to $t = 0.5$ seconds in steps of $\triangle t = 0.0002$ seconds.

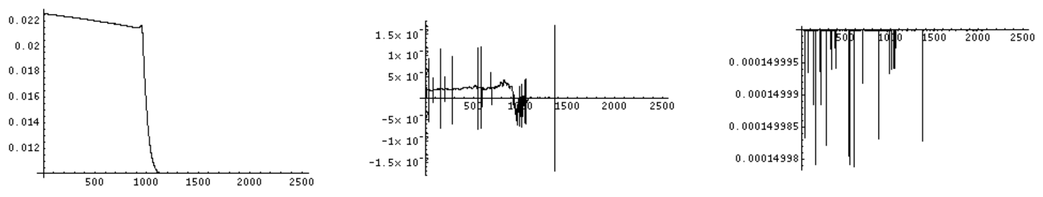
The bottom of the container is a no-slip boundary condition. At the side we have slip boundary conditions. At the top we have an outlet and specify the pressure $p = 0\,\mathrm{Pa}$. The result at different times can be seen in Figure 8.10. Again, the driving force for the bubble to rise is the difference in density between water and oil despite the counter action of gravity which was set to $g = 9.82\,{}^{\mathrm{m}}/_{\mathrm{s}^2}$.


(a) $t = 0.15s$ (b) $t = 0.175s$ (c) $t = 0.185s$ (d) $t = 0.19s$

(e) $t = 0.2s$ (f) $t = 0.21s$ (g) $t = 0.245s$ (h) $t = 0.3s$

Figure 8.10: The sequence of pictures shows a rising oil bubble in a water tank. Initially the oil is rising without significantly changing it's shape. Shortly under the upper oil's surface the water layer bursts and snaps back and the oil bubble merges with the oil in the upper part of the tank.

As the bubble rises the bubble's shape remains circular because of the surface tension σ and the oil's large viscosity. The bursting of the bubble occurs at time $t = 0.18\,\mathrm{s}$ and agrees very well with the results returned by other simulation software [3]. In our case the static equilibrium is reached at time time $t = 0.3\,\mathrm{s}$ which is much earlier then in the reference simulation [3] where equilibrium is reached at $t = 0.5\,\mathrm{s}$. In Figure 8.11 we see the monitoring quantities of the simulation. The length of the interface, the amount the level set is corrected and the area conservation in each time step. The total area occupied by oil can be computed to be $a \hat{=} 0.01 \cdot (0.015 - 0.01) + \pi \cdot 0.002^2 \hat{=} 0.0625664 \cdot 10^{-3}\,\mathrm{m}^2$. This implies that the height of the water in steady state should be at $h \hat{=} 0.015 - (0.0625664 \cdot 10^-3)/0.01 \hat{=} 0.00874336\,\mathrm{m}$. The measured height is $0.00874907\,\mathrm{m}$ which corresponds to a 0.065% higher water line in steady state.



(a) Length of the interface (b) Level-set correction (c) Area conservation

Figure 8.11: Depicted are the length of the interface in meter, the level-set correction in meter and the area conservation in square meters for the rising oil bubble in water. On the x-axis we show the simulation's time steps. If we consider the negative implicit function, which represents oil in this case, before merging, the degree of convexity is higher than that of concavity. This results in a loss of mass (see also Figures 6.8 and 6.9) which is counteracted by the positive level-set correction until about simulation step 1000 which corresponds to $t = 0.2\,\mathrm{s}$. Once the degree of concavity supersedes the degree of convexity the level-set correction is negative.

It is, however, noteworthy, that the length of the interface Γ decreases until the merging with the top oil line. This implies that some material - even though the total amount is preserved well - is shifted from the inside of the bubble to the top.

8.3 The Ink-Jet Model

As a last model we present an ink-jet print head. We will model the ejection of ink, the flight of the droplet as well as the impact of the droplet onto the solid wall target. Accordingly the geometry of the ink-jet simulation domain is made up of three parts. First we have a nozzle, second a flight chamber and third the target area. Figure 8.12 shows the details of the geometry. For a drop on demand ink-jet typical outlet diameters of a nozzle are in the range of $20 - 70 \cdot 10^{-6}\,\mathrm{m}$. The initial velocities are in a range of $1 - 5\,{}^{\mathrm{m}}\!/\!_{\mathrm{s}}$[11, 16, 17].

For the ink-jet model we make the Navier-Stokes equation non dimensional [17, 16]. The dimensionless Navier-Stokes equation is written as:

$$\begin{aligned} \rho\frac{\partial}{\partial t}\mathbf{u} + \rho\left(\mathbf{u}\cdot\nabla\right)\mathbf{u} - \frac{1}{Re}\nabla\mu\nabla\mathbf{u} + \nabla p &= \frac{1}{We}\kappa(\phi)\delta(\phi)\nabla\phi \\ \nabla\cdot\mathbf{u} &= 0. \end{aligned}$$

Here Re is the Reynolds number and We is the Weber number defined as:

$$Re = \tfrac{\rho_1 U L}{\mu_1} \quad \text{and} \quad We = \tfrac{\rho_1 U^2 L}{\sigma}\ ,$$

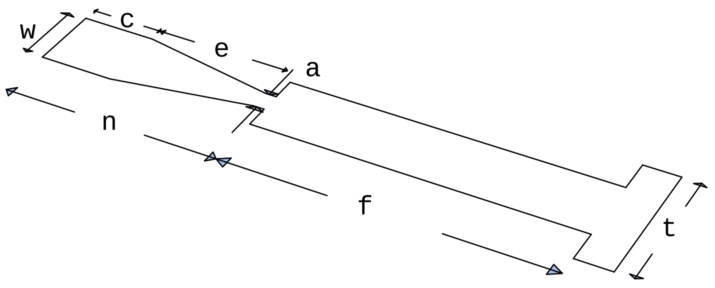


Figure 8.12: The geometry of the ink-jet model. The model is made up of three parts, the nozzle, the fight chamber and the target area and has a total length of $1.6\,10^{-3}\,\mathrm{m}$. The nozzle n has a total length of $0.6\,10^{-3}\,\mathrm{m}$. The length of the flight chamber f measures $0.9\,10^{-3}\,\mathrm{m}$. The rest of $0.1\,10^{-3}\,\mathrm{m}$ belongs to the target area. The target area t has a width of $0.4\,10^{-3}\,\mathrm{m}$ and the nozzle w a is $0.2\,10^{-3}\,\mathrm{m}$ wide. The nozzle itself consists of three parts. The ink chamber c which is $0.2\,10^{-3}\,\mathrm{m}$ long and a cone e which has a length of $0.37\,10^{-3}\,\mathrm{m}$. The cone narrows down to the opening of the nozzle a which is $0.06\,10^{-3}\,\mathrm{m}$ wide. The aperture and third part of the nozzle is the remaining $0.03\,10^{-3}\,\mathrm{m}$ long.

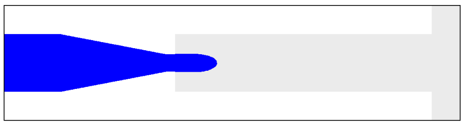
respectively. This allows us to set the following:

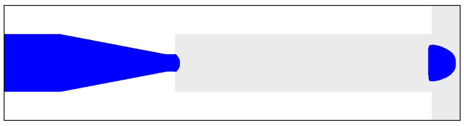
$$\begin{aligned}\rho(\phi) &= \begin{cases} 1 & \text{if } \phi \geq 0, \\ \rho_2/\rho_1 & \text{if } \phi < 0, \end{cases} \\ \mu(\phi) &= \begin{cases} 1 & \text{if } \phi \geq 0, \\ \mu_2/\mu_1 & \text{if } \phi < 0. \end{cases}\end{aligned}$$

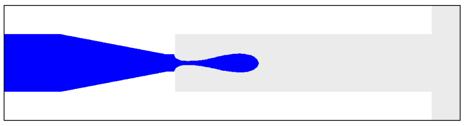
The primary fluid (index 1) is a dye-*based* ink. The dye in the ink is completely dissolved. Because of this dye-based inks flow better than for example pigment based inks. The choice of a dye-based ink implies we are expecting Newtonian fluid behaviour. The surrounding fluid is air (index 2). The chosen physical parameters [3] are:

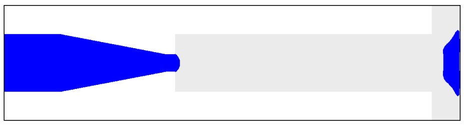
$$\begin{aligned}\rho_1 &= 1000\,{}^{\mathrm{kg}}/_{\mathrm{m}^3}, & \rho_2 &= 1.225\,{}^{\mathrm{kg}}/_{\mathrm{m}^3} \\ \mu_1 &= 1.\cdot 10^{-2}\,\mathrm{Pa\,s}, & \mu_2 &= 1.77894\cdot 10^{-5}\,\mathrm{Pa\,s} \\ \sigma &= 0.07\,{}^{\mathrm{N}}/_{\mathrm{m}}, & g &= 0\,{}^{\mathrm{m}}/_{\mathrm{s}^2}.\end{aligned}$$

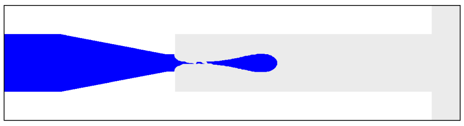
We take the typical velocity to be $U = 5\,{}^{\mathrm{m}}/_{\mathrm{s}}$ and the typical length as $L = 30\cdot 10^{-6}\,\mathrm{m}$. This results in a Reynolds number $Re = 15$ and a Weber number $We = 10.7143$. The nozzle has no-slip boundary conditions set. The inflow profile is parabolic with a maximum of $u_{max} = 1.25\,{}^{\mathrm{m}}/_{\mathrm{s}}$. This u_{max} is smoothly ramped up in $2\,\mu\mathrm{S}$, held for $10\,\mu\mathrm{S}$ and them smoothly ramped down in $2\,\mu\mathrm{S}$. This input velocity model results from a pressure peak at $0\,\mu\mathrm{S}$ and a negative pressure peak at $10\,\mu$S[12]. The target is modelled with slip boundary conditions. All other boundaries are outlet boundaries. The interface thickness is set to $h_w = 1$ and full re-initialisation is done after each time step. The simulation was performed from $t_0 = 0\,\mu$S to $t_e = 150\,\mu$S in steps of $\triangle t = 0.5\,\mu$S. The mesh consisted of 5725 elements which corresponds to 38389 degrees of freedom. The result at various steps in time can be seen in Figure 8.13.

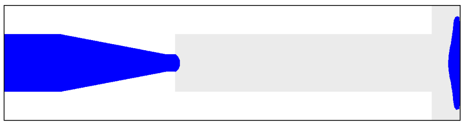

(a) $t = 10\mu S$

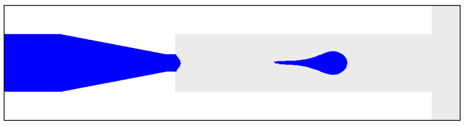

(b) $t = 85\mu S$

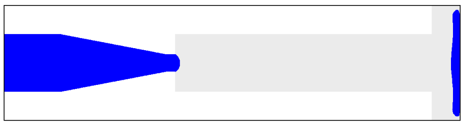

(c) $t = 20\mu S$

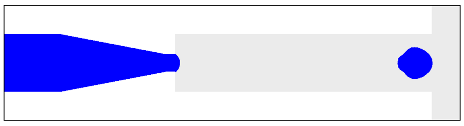

(d) $t = 95\mu S$


(e) $t = 25\mu S$


(f) $t = 105\mu S$


(g) $t = 45\mu S$


(h) $t = 115\mu S$


(i) $t = 75\mu S$

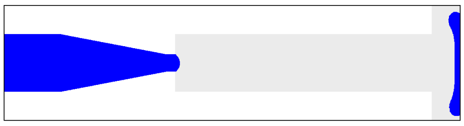

(j) $t = 150\mu S$

Figure 8.13: The figure shows an ink-jet ejection at various phases. In the first $2\,\mu$S the inflow velocity is ramped up to it's maximum of $1.25\,{}^{\mathrm{m}}/_{\mathrm{s}}$. The inflow profile is held for another $10\,\mu$S after which it is ramped down in $2\,\mu$S. At $t = 10\,\mu$S we observe an ink column which is thins out after $t = 12\,\mu$S. Pinch off is approximately at $t = 25\,\mu$S. The surface tension contracts the droplet's tail and forms a sphere. The momentum of the contraction creates an ellipsoidal droplet just before impact. Impact on the target is at about $t = 86\,\mu$S. After impact the droplet spreads out and contracts again slightly.

After the inflow velocity's ramp up phase is completed at time $t = 2\,\mu$S, the droplet ejects from the nozzle opening in straight column until the inflow velocity is ramped down at $t = 12\,\mu$S. The ink column then begins thinning out and the single droplet pinches off at approximately $t = 25\,\mu$S. In this configuration we see that the tail of the droplet is sucked up into the main droplet. No satellite droplet that later hits the leading droplet forms. After impact, which occurs at time $t = 86\,\mu$S, we observe a spreading on the surface. The droplet develops a doughnut form. Also, due to a lack of a contact angle model the meniscus at the opening of the nozzle is outward formed. To some degree this could be overcome by applying a backward pull on the inlet boundary condition. Nevertheless, this does not circumvent the need for a contact angle model to accurately model the ink at the out flow of the nozzle. It is remarkable that the droplet on the target is so well formed even without a contact angle model.

8.4 Final Remarks

8.4.1 Conclusions

Unlike many existing highly specialised simulation software packages a unified approach for the modelling based on ordinary and partial differential equations has been presented. This unified approach is based on a simple operator concept.

An outline of how to generally model lumped systems based on Lagrange's method and lumped operators has been given and adopted for computer use. Electrical circuits have been used as exemplary lumped systems and the software has been designed in a manner which allows for extending the modules to other lumped system domains.

A general purpose non-linear coupled partial differential equation solving environment has been presented. The environment includes techniques for generally linearising second order in space and time partial differential equations. We linearised the Navier-Stokes equation and coupled the linearised equation to a level-set method to model free surface flow.

The implemented partial differential equations operators are based on the finite element method which has been derived in great detail. Based on this detailed derivation we presented a set of general purpose partial differential operators. The implemented operators can be used for numerical computations including machine or arbitrary precision numbers, real, integer or imaginary complex numbers. Additionally the operators can handle symbolic input which may be used to derive symbolic finite elements.

We presented sophisticated algorithms for both, solving systems non-linear equations via the affine invariant Newton method and differential algebraic equations via the backward difference formulae. These algorithms have been chosen for the wide range of applications they may be used in.

Based on the above we have implemented a Navier-Stokes solver which is coupled to a level-set method to model free surface flow. The algorithm is flexible enough to solve a wide range of fluid flow and free surface problems. It allows for flows with merging and breaking interfaces and handles discontinuities in density and viscosity. The surface tension forces were also accounted for in the simulations. Last, an ink-jet model has been presented.

8.4.2 Future work

From a general perspective the following topics could be addressed to further generalise the present work:

1. Graphs containing graphs
   The ability to allow a graph to contain a sub-graph would be helpful to create libraries of e.g. lumped systems of high complexity and re-use them.
2. Proper Neumann handling boundary integrals
   In the implementation of the partial differential equation operators based on the finite element method a proper implementation of the boundary part of the operator would allow for a more flexibility in the choice of the Neumann boundary conditions.
3. Adaptive step size and automatic order selection for the backward difference formulae (BDF) time integrator
   An extension in this direction would allow for a more autonomous algorithm in the sense that the user must not care too much about selecting an appropriate time step.
4. Access to the operator internal data
   Each operator needs an input function to fully specify the operators behaviour. Sometimes it is necessary to, for example, model the derivative of the unknown function. Currently, this has to be done by interpolating the unknown function in the appropriate element and then constructing the derivative of that interpolation and then feed the result back into the operator. The operator has, however, internally available all information to compute the derivative. It would advantageous if it were possible to tell the operator to make use of derivative information and thus not need to construct the derivative externally. This procedure would also be beneficial for other mathematical constructs than the derivative.

To further refine the presented free surface model several directions are possible:

1. Modelling the contact angle
   In this work no contact angle model has been applied. For a more realistic

simulation an implementation of a suitable model is advisable. The impact of not using a contact angle model is seen in the way the liquid behaves at the nozzle after ejection and the droplet after impact.

2. Merging of the Navier-Stokes step with the advection of the implicit function and the re-initialisation.
   In order to maintain quadratic convergence of the nonlinear solver the linearisation of both the Navier-Stokes and level set equation have to be performed adequately. For the Navier-Stokes equation this has been done in the current work. The level-set equation and the re-initialisation, however, are two separate equations. At least the level-set equation could, in its current form, be directly coupled to the Navier-Stokes equation. This however, would reduce the quadratic convergence to a linear convergence. A unification of the re-initialisation and the level-set method and a coupling to the Navier-Stokes equation are necessary.

3. Avoid the dependence on the interface thickness h_w
   If the dependence on additional control parameters can not be avoided, at least a good function for estimating a good h_w were very useful.

4. Adaptive mesh refinement
   Experiments in this direction have been conducted. The current structure of coupling the Navier-Stokes equation and the level-set method is done in successive steps. Even though each of these time integration steps is performed implicitly, applying a h-adaptivity strategy makes the procedure explicit. This assumption is supported by the observation that once an adaptive strategy is applied the resulting velocity field is oscillating. Even for very small time steps the oscillation can not be overcome and thus making an adaptive strategy for this setup unprofitable. In future work the first step could be to make the coupling and re-initialisation fully implicit and then apply adaptivity.
   Another possible way to increase the accuracy is to employ quadrilateral elements in stead of the triangular elements.

References

[1] Uri M. Ascher and Linda R. Petzold. *Computer Methofs for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, 1998.

[2] R. Clift, J. R. Grace, and M. E. Weber. *Bubbles, Drops and Particles.* Academic Press, 1978.

[3] COMSOL. *COMSOL User's Guide 3.3.*

[4] C.W. Gear and L.F. Shampine. A user's view of solving stiff ordinary differential equations. *SIAM Review*, 21, No.1:1 – 17, 1979.

[5] Michael Hanke. A new implementation of a bdf method within the method of lines. *J. Comput. Meth. Sci. Eng.*, To appear 2003.

[6] Michael T. Heath. *Scientific Computing An Introductory Survey*. McGraw Hill, 2002.

[7] B. Lafaurie, C. Nardone, R. Scardovelli, S. Zaleski, and Zanetti G. Modelling merging and fragmentation in multiphase flow with surfer. *Journal of Computational Physics*, 113:134 – 147, 1994.

[8] S. Nagrath, K. E. Jansen, and Lahey Jr. R. T. Computation of incompressible bubble dynamics with a stabilized finite element level set method. *Computer methods in applied mechanics and engineering*, 194:4565 – 4587, 2005.

[9] Nowak and Weimann. A family of newton codes for systems of highly nonlinear equations. *Konrad-Zuse-Zentrum*, Technical Report TR-91-10:na, 1991.

[10] G. Opfer. *Numerische Mathematik für Anfaenger*. Vieweg, 2001 (3rd Ed.).

[11] Nuno Reis, Chris Ainsley, and Brian Derby. Ink-jet delivery of particle suspensions by piezoelectric droplet ejectors. *J. Appl. Phys.*, 97:094903, 2005.

[12] T. W. Shield, D. B. Body, and F. E. Talke. Drop formation by dod ink-jet nozzles: A comparison of experiment and numerical simulation. *IBM Journal of Research and Development*, 31(1):96–110, 1987.

[13] Anton Smolianski. Finite-element/level-set/opereator-splitting (felsos) approach for computing two-fluid unsteady flows with free moving interfaces. *International Journal for Numerical Methods in Fluids*, Volume 48, Issue 3:231 – 269, 2004.

[14] Albert Y. Tong and Zhaoyuan Wang. A numerical method for capillarity-domiant free surface flows. *Journal of Computational Physics*, 221(Issue 2):506–523, 2007.

[15] G. Wanner and E. Hairer. *Solving Ordinary Differential Equations II Stiff and Differential-Algebraic Problems*. Springer, 1996.

[16] Jiun-Der Yu, Shinri Sakai, and James Sethian. A coupled level set projection method applied to ink jet simulation. *Interfaces and Free Boundaries*, 5:459 – 482, 2003.

[17] Jiun-Der Yu, Shinri Sakai, and James Sethian. A coupled quadrilateral grid level set projection method applied to ink jet simulation. *Journal of Computational Physics*, 206:227 – 251, 2005.

9 Appendix

9.1 Quadratic Serendipity Shape Functions

$$\begin{array}{l}
\frac{1}{8}\left((r+1)(s+1)(t+1)-\left(1-r^2\right)(s+1)(t+1)-(r+1)\left(1-s^2\right)(t+1)-(r+1)(s+1)\left(1-t^2\right)\right)\\
\frac{1}{8}\left((1-r)(s+1)(t+1)-\left(1-r^2\right)(s+1)(t+1)-(1-r)\left(1-s^2\right)(t+1)-(1-r)(s+1)\left(1-t^2\right)\right)\\
\frac{1}{8}\left((1-r)(1-s)(t+1)-\left(1-r^2\right)(1-s)(t+1)-(1-r)\left(1-s^2\right)(t+1)-(1-r)(1-s)\left(1-t^2\right)\right)\\
\frac{1}{8}\left((r+1)(1-s)(t+1)-\left(1-r^2\right)(1-s)(t+1)-(r+1)\left(1-s^2\right)(t+1)-(r+1)(1-s)\left(1-t^2\right)\right)\\
\frac{1}{8}\left((r+1)(s+1)(1-t)-\left(1-r^2\right)(s+1)(1-t)-(r+1)\left(1-s^2\right)(1-t)-(r+1)(s+1)\left(1-t^2\right)\right)\\
\frac{1}{8}\left((1-r)(s+1)(1-t)-\left(1-r^2\right)(s+1)(1-t)-(1-r)\left(1-s^2\right)(1-t)-(1-r)(s+1)\left(1-t^2\right)\right)\\
\frac{1}{8}\left((1-r)(1-s)(1-t)-\left(1-r^2\right)(1-s)(1-t)-(1-r)\left(1-s^2\right)(1-t)-(1-r)(1-s)\left(1-t^2\right)\right)\\
\frac{1}{8}\left((r+1)(1-s)(1-t)-\left(1-r^2\right)(1-s)(1-t)-(r+1)\left(1-s^2\right)(1-t)-(r+1)(1-s)\left(1-t^2\right)\right)\\
\frac{1}{4}\left(1-r^2\right)(s+1)(t+1)\\
\frac{1}{4}(1-r)\left(1-s^2\right)(t+1)\\
\frac{1}{4}\left(1-r^2\right)(1-s)(t+1)\\
\frac{1}{4}(r+1)\left(1-s^2\right)(t+1)\\
\frac{1}{4}\left(1-r^2\right)(s+1)(1-t)\\
\frac{1}{4}(1-r)\left(1-s^2\right)(1-t)\\
\frac{1}{4}\left(1-r^2\right)(1-s)(1-t)\\
\frac{1}{4}(r+1)\left(1-s^2\right)(1-t)\\
\frac{1}{4}(r+1)(s+1)\left(1-t^2\right)\\
\frac{1}{4}(1-r)(s+1)\left(1-t^2\right)\\
\frac{1}{4}(1-r)(1-s)\left(1-t^2\right)\\
\frac{1}{4}(r+1)(1-s)\left(1-t^2\right)
\end{array}$$

9.2 Transient Mechanics

The equation to model transient stress and strain of solids is given by 4.8 with $\tau_2 = \gamma = 0$.

$$\begin{pmatrix}
\frac{Yu_{zz}}{2(\nu+1)}+\frac{Yu_{yy}}{2(\nu+1)}+\frac{Y(1-\nu)u_{xx}}{(1-2\nu)(\nu+1)} & \frac{Yv_{xy}}{2(\nu+1)}+\frac{Y\nu v_{xy}}{(1-2\nu)(\nu+1)} & \frac{Yw_{xz}}{2(\nu+1)}+\frac{Y\nu w_{xz}}{(1-2\nu)(\nu+1)} \\
\frac{Yu_{xy}}{2(\nu+1)}+\frac{Y\nu u_{xy}}{(1-2\nu)(\nu+1)} & \frac{Yv_{zz}}{2(\nu+1)}+\frac{Y(1-\nu)v_{yy}}{(1-2\nu)(\nu+1)}+\frac{Yv_{xx}}{2(\nu+1)} & \frac{Yw_{yz}}{2(\nu+1)}+\frac{Y\nu w_{yz}}{(1-2\nu)(\nu+1)} \\
\frac{Yu_{xz}}{2(\nu+1)}+\frac{Y\nu u_{xz}}{(1-2\nu)(\nu+1)} & \frac{Yv_{yz}}{2(\nu+1)}+\frac{Y\nu v_{yz}}{(1-2\nu)(\nu+1)} & \frac{Y(1-\nu)w_{zz}}{(1-2\nu)(\nu+1)}+\frac{Yw_{yy}}{2(\nu+1)}+\frac{Yw_{xx}}{2(\nu+1)}
\end{pmatrix}$$

We have sub matrices of size 3x3 (3 space dimensions) and there are 3x3 sub matrices since we deal with 3 unknown functions (u,v and w) and we set $a = \frac{Y(1-\nu)}{(1-2\nu)(\nu+1)}$, $b = \frac{Y\nu}{(1-2\nu)(\nu+1)}$ and $c = \frac{Y}{2(\nu+1)}$:

$$\begin{pmatrix} \begin{pmatrix} a & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & c \end{pmatrix} & \begin{pmatrix} 0 & b & 0 \\ c & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & b \\ 0 & 0 & 0 \\ c & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & c & 0 \\ b & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} c & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & c \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & b \\ 0 & c & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & c \\ 0 & 0 & 0 \\ b & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & c \\ 0 & b & 0 \end{pmatrix} & \begin{pmatrix} c & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & a \end{pmatrix} \end{pmatrix}$$

9.3 Acknowledgements

First things first: Hi Mom!

Writing this work was supported and made possible by many people. Here I wish to express my gratitude to them. I wish to thank:

Prof. Jan Korvink for giving me the opportunity to work in his group in a friendly and prolific atmosphere.

Prof. Peter Woias for taking the time and interest in co-examining this thesis.

Zhenyu Liu, for carefully guiding me and having the patience to let me explore my ideas and learn and being ready with new solution approaches when my ideas miserably failed.

All my colleagues in the group for your patience, friendship and all your help. Don't you ever clean that coffee machine. Special gratitude goes to my colleagues Christian Moosmann and Jan Lienemann. We fought the Frontend and the Kernel in many battles - we prevailed. I really enjoyed working with you.

The people at IMTEK and all over the world contributing in so many ways to the IMS. In fact there were so many contributions ranging from motivating emails to improvement ideas to code contributions, that it is impossible to mention everybody. I have tried to maintain a list of contributors which is part of the IMS distribution. Thank you all!

The people reading and making invaluable suggestions of what texts worked for them and which did not were Eldad Louw, Christian Moosmann and Zhenyu Liu. They have read the manuscript again and again and their suggestions made a great difference.

Anne Rotler & Alice Koopmann, you were carefully watching over me while I was asleep.

My family and my friends for their love - it is invaluable to know that I can always count on you. This feeling is an infinite resource of strength.

Wendy Kreß, for introducing me to your way of cutting a cake. Thank you for your love that made me perceive the world in a different, much richer light.